Titre: Title:	Conception d'un processeur à vitesse variable et synthèse d'horloge à période ajustable
Auteur: Author:	Dimitri Gabriel Epassa Habib
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Epassa Habib, D. G. (2006). Conception d'un processeur à vitesse variable et synthèse d'horloge à période ajustable [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7709/

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UN PROCESSEUR À VITESSE VARIABLE ET SYNTHÈSE D'HORLOGE À PÉRIODE AJUSTABLE

EPASSA HABIB DIMITRI GABRIEL

DÉPARTEMENT DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17940-6
Our file Notre référence
ISBN: 978-0-494-17940-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CONCEPTION D'UN PROCESSEUR À VITESSE VARIABLE ET SYNTHÈSE D'HORLOGE À PÉRIODE AJUSTABLE

présenté par : EPASSA Habib Dimitri Gabriel

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doct., Présidente

M. BOYER François-Raymond, Ph.D., membre et directeur de recherche

M. SAVARIA Yvon, Ph.D., membre et codirecteur de recherche

M. ABOULHAMID El Mostapha, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier Messieurs Francois-Raymond Boyer et Yvon Savaria, respectivement directeur et co-directeur de maîtrise pour leur disponibilité, leurs conseils et suggestions, ainsi que le support financier qu'ils ont apporté afin de mener à bien ce projet de recherche jusqu'à sa consécration.

Mes remerciements vont aussi à tous les collègues et camarades du GRM (Groupe de Recherche en Microélectronique) sans exception, des étudiants aux techniciens et personnel administratif. En un mot, tous ceux qui ont fait partie de mon quotidien durant ces trois ans d'études supérieures. En particulier, à M. Bill Pontikakis qui a partagé avec moi les tâches au tout début du projet.

Une mention spéciale à mes parents, et ma famille étendue du Cameroun au Canada, pour leur présence et leur soutien moral tout au long de mes études. Sans oublier, mes amis qui m'ont permis de m'épanouir en dehors du milieu estudiantin.

RÉSUMÉ

Avec l'avancée technologique des produits électroniques portables, les manufacturiers essaient du mieux qu'ils peuvent de se rapprocher des performances des ordinateurs de bureaux. Ces derniers, puisant leur source d'énergie directement des prises du réseau électrique domestique, ne sont pas confrontés aux mêmes problèmes sévères d'économie d'énergie. En revanche, les équipements portatifs fonctionnant à l'aide de batteries ont besoin d'une autonomie satisfaisante pour les usagers tout en maintenant un niveau de performance suffisant. Étant donné que la performance est proportionnelle à la consommation d'énergie, un compromis s'impose lors du design des processeurs actuels. La majeure partie de la recherche, concernant les manufacturiers populaires comme Intel, AMD, IBM, etc. est axée sur la conception de processeurs embarqués à consommation d'énergie réduite.

Plusieurs méthodes sont employées pour la conception de tels processeurs, la plupart allient l'ajustement de la tension d'alimentation avec la variation de la fréquence de l'horloge système, méthode utilisée par les processeurs DVS (Dynamic Voltage Scaling). Dans ce rapport, les différentes architectures et méthodes de réduction de l'énergie consommée dans les processeurs embarqués sont explorées. Ces derniers utilisent des circuits de génération d'horloge dynamiquement ajustables ou synthétiseurs d'horloge programmables.

Nous présenterons dans ce mémoire la conception d'un processeur embarqué à vitesse variable (VSP) dont la période de l'horloge est ajustée à chaque cycle, selon les instructions qui se présentent pour exécution dans le pipeline. Pour atteindre nos objectifs, le projet de recherche est effectué en deux volets : En premier lieu, la conception du VPCS, un circuit de synthèse d'horloge à période d'horloge variable, est

effectuée en technologie ASIC 0,18µm. Le circuit permet d'effectuer la multiplication ainsi que la division de la fréquence de l'horloge de référence par un facteur fractionnel quelconque, contrairement au DDPS de la littérature existante qui ne permet que de multiplier la fréquence par un facteur plus grand que un. Le circuit proposé suit les règles de conception des circuits à consommation réduite de puissance, et est comparable en performance aux circuits de la littérature. Il se présente sous forme d'un noyau logiciel codé en VHDL facilement intégrable avec tout processeur dont le noyau serait accessible.

En second lieu, le processeur proposé est basé sur un processeur RISC standard de type Nios provenant de la société Altera®. En combinaison avec le VPCS et modifié pour obtenir les fonctionnalités désirées, il est implémenté dans la plateforme de systèmes embarqués du FPGA Stratix™. Un prototype est réalisé avec succès, fonctionnant avec une horloge dont la fréquence oscille entre 110 MHz et 147 MHz selon que les instructions soient « lentes » ou « rapides », permettant d'obtenir une accélération par rapport au Nios standard à performance maximale avec une consommation d'énergie par opération bien réduite.

ABSTRACT

With the technology advances applied to electronic portable devices, the main goal for their manufacturers is to achieve as much as possible the same performance as desktop personal computers. With the latter, the energy source is taken from domestic electric outlet and there is no pressing need to save energy, while battery-powered portable devices need long lasting battery life, keeping a sufficient level of performance. Since having an increase in performance leads to an increase in energy consumption, energy efficient processors design becomes a very pressing issue. Manufacturers of popular processors such as Intel, AMD, IBM, etc. address that issue with some commercial embedded processors designed to be low power.

Many methods exist to design the above processors. The major methods will achieve energy reduction by adjusting the supply voltage along with the operating frequency which is used in Dynamic Voltage Scaling (DVS) processors. In this thesis, we explore different methods and architectures for energy reduction in embedded processors, which use dynamic adjustable clock generators or programmable frequency synthesizers.

We present a Variable Speed Processor (VSP), which is an embedded processor with a system clock period that can change every cycle, according to the instructions being executed by the pipeline. To achieve our goal, the design process is done in two steps. First, we designed a Variable Period Clock Synthesis Circuit (VPCS) in ASIC 0,18μm technology, which can multiply and divide on the fly the reference clock frequency with a fractional number that can be larger than unity, compared to the Direct Digital Period Synthesis Circuit (DDPS) that can only multiply the frequency (multiply the period by a number below unity). The proposed circuit design follows the low power design rules and is comparable to clock generators found in the literature in terms of performance.

Secondly, the proposed processor is implemented by coupling the VPCS and the standard RISC Nios processor from Altera. The Nios core is modified so that the desired functionalities are achieved. The design is prototyped into the embedded systems platform for the Stratix™ FPGA. The experimental circuit runs with a 110 MHz frequency for "slow" instructions and 147 MHz for "fast" instructions, giving a speedup upon the standard Nios at peak performance with energy per operation savings.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

SIGLES ET ABRÉVIATIONS

ADPLL	: All Digital PLL
CPU	: Central Processing Unit
CVS	: Clustered Voltage Scaling
DCO	: Digitally Controlled Oscillator
DDPS	: Direct Digital Period Synthesis
DDS	: Direct Digital Synthesis
DFCS	: Dynamic Frequency Clocking Scheme
DPCD	: Dynamic Clock Programmable Divider
DPM	: Dynamic Power Management
DVS	: Dynamic Voltage Scaling
PIO	: Parallel Interface Output
PLL	: Phase Locked Loop
SAIF	: Switching Activity Input File
SDF	: Standard Delay File
UART	: Universal Asynchronous Receiver Transmitter
VCO	: Voltage Controlled Oscillator
VSP	: Variable Speed Processor

LISTE DES ANNEXES

CHAPITRE 1
Introduction

1.1 Motivations

Les designs des processeurs actuels ne se consacrent pas seulement aux besoins de haute performance à combler, mais de plus en plus, ils répondent à une nécessité de dissiper le moins de chaleur possible, c'est-à-dire de consommer moins de puissance que les processeurs employés dans les ordinateurs personnels de bureau. Il s'agit aussi de diminuer la consommation de l'énergie provenant des piles ou des batteries, pour ce qui est des processeurs d'équipements embarqués tels que les téléphones cellulaires, les organiseurs numériques personnels, etc.. L'idée première qui a orienté la rédaction de ce mémoire était de vérifier dans quelle mesure un circuit électronique digital, qui s'adapterait aux conditions variables de fonctionnement, serait implémenté. Cette variabilité pourrait venir des changements de température du circuit, des variations des procédés de fabrication, des changements de la tension d'alimentation ainsi que des variations de la source d'énergie. Ce circuit réduirait ainsi dynamiquement sa vitesse au fur et à mesure que la température augmente, ou lorsque les délais du circuit s'allongent, ou encore quand la tension ou l'énergie restante dans la batterie baissent. Les grands constructeurs de processeurs tels que Intel® et AMD® intègrent déjà ces astuces, fabricant ainsi des processeurs alliant haute performance et faible consommation.

1.2 Objectifs de la recherche

Faire varier la vitesse d'opération selon les conditions de fonctionnement est un facteur clé dans la conception de processeurs à faible consommation de puissance et d'énergie. Plusieurs circuits dans la littérature courante sont capables de synthétiser l'horloge à

partir d'une horloge de référence. Cette fonction est paramétrable, donc la cadence de l'horloge peut être ajustée selon les besoins du concepteur.

L'objectif principal du projet réalisé tout au long de ce mémoire est de concevoir un circuit électronique qui change dynamiquement sa période d'horloge dans un intervalle d'un seul cycle d'horloge.

Nos recherches ont conduit à l'ajout de plusieurs fonctions aux circuits existants dans la littérature, en concevant un circuit de synthèse d'horloge à période variable aussi dynamique, complètement numérique et intégrable sur puce que le DDPS, mais avec une couverture de fréquence plus élevée vue sa capacité à diviser et multiplier la période de l'horloge.

Les applications courantes les plus en demande de variabilité de la fréquence des opérations sont les processeurs à consommation de puissance faible et ajustable selon la charge de travail. Le projet de recherche vise, par la même occasion, à fabriquer un tel processeur en s'aidant du circuit de synthèse d'horloge à période variable, avec une méthodologie compatible aux architectures à faible consommation de puissance existantes, mais différentes du point de vue de l'ajustement de la fréquence d'opération.

1.3 Contributions et publications

Ce mémoire contient les fruits d'une recherche avancée qui ont été synthétisés dans la production de 2 articles de conférence et d'un article soumis auprès d'une revue scientifique.

Le premier article présente la conception du circuit de VPCS, le générateur d'horloge à période variable adapté aux circuits de faible consommation de puissance. Le second article présente un processeur embarqué à vitesse variable utilisant le VPCS pour ajuster la période de son horloge selon chaque instruction issue de son pipeline prête à être exécutée. Le troisième article, soumis pour publication dans une revue présente le processeur à vitesse variable avec son générateur d'horloge à période variable en détails, et effectue une analyse de la performance ainsi que de la réduction en consommation d'énergie atteinte.

Articles de Conferences:

1 - Boyer, F-R.; Epassa, H.G.; Pontikakis, B.; Savaria, Y.; Wei Ling, - "A variable period clock synthesis (VPCS) architecture for next-generation power-aware SoC applications"
Circuits and Systems, 2004. NEWCAS 2004. The 2nd Annual IEEE Northeast Workshop on, 20-23 June 2004 Page(s):145 - 148

2 - Epassa, H.G; Boyer, F-R; Savaria, Y – "Implementation of a Cycle by Cycle Variable Speed Processor" Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on 23-26 May 2005 Page(s):3335 - 3338

Article de Revue:

Boyer, F-R; Epassa, H.G; Savaria, Y – "Embedded Power-Aware Cycle by cycle Variable Speed Processor"
Soumis pour publication au *IEE Proceedings Computers & Digital Techniques*

Circuits de prototypage : Carte de développement de système embarqué FPGA Stratix™ EPS140

Le circuit a été implémenté sur la plateforme de prototypage rapide « *Embedded System development platform, Stratix Edition* » de la société Altera qui contient un processeur RISC Nios dont le noyau logiciel est disponible et modifiable.

1.4 Plan du mémoire

Ce mémoire est composé de cinq chapitres répartis comme suit :

Le chapitre 2 présente une revue de littérature sur l'évolution de la synthèse d'horloge à l'aide de PLL, incluant les circuits de synthèse numérique d'horloge récemment utilisés, ainsi que les différentes architectures de processeurs à basse puissance, utilisant les techniques de réduction de la consommation d'énergie et de puissance.

Le chapitre 3 élabore sur les méthodes de conception qui ont contribué à la réalisation de ce projet. Ce chapitre décrit de manière détaillée, la méthodologie et les outils utilisés spécifiquement lors de la conception d'un processeur à vitesse variable sur une plateforme FPGA avec un processeur embarqué.

Le chapitre 4 présente les résultats issus de notre travail. Il s'agira d'une compilation des travaux effectués tout au long du projet, allant de la conception du générateur d'horloge VPCS à l'implémentation du VSP. On y retrouve également des résultats d'implémentation, une étude détaillée et comparative de la méthode à période d'horloge variable appliquée au processeur embarqué NIOS. Le chapitre est un condensé de l'article de revue, soumis pour publication auprès du journal « *Computer and Digital techniques* » de la société IEE.

Enfin, en guise de conclusion, le dernier chapitre fait un bilan du travail de recherche effectué et propose des ouvertures éventuelles pour des travaux futurs.

CHAPITRE 2
Revue de littérature

Dans les systèmes à microprocesseurs, on retrouve toujours une partie de circuit dédiée à la génération de l'horloge du système. Ce type de circuiterie réalise ce qui est communément appelé une synthèse de fréquence. L'horloge provenant du cristal de quartz est retransformée pour rencontrer les spécifications requises pour les circuits numériques synchrones, ainsi que pour les réseaux de distribution d'horloge. Ce chapitre explore les différentes architectures de circuits de synthèse d'horloge existants, ainsi que les différentes architectures de processeurs embarqués à basse puissance, dont la fréquence de l'horloge est dynamiquement variable. Ceci permettra de faire une comparaison entre ces architectures et la notre.

2.1 Circuits de synthèse d'horloge

Depuis la découverte des propriétés oscillatoires des cristaux, les travaux sur les circuits de synthèses d'horloge se sont intensifiés. Ce paragraphe présente ces circuits en partant du traditionnel PLL jusqu'aux circuits de synthèse d'horloge actuels.

2.1.1 Synthèse d'horloge traditionnelle : les PLL

Il y a environ un siècle, l'effet piézoélectrique des cristaux fut découvert par les chercheurs Pierre et Jacques Curie. En effet, en appliquant un champ électrique autour d'un cristal, sa structure subit une distorsion, lui donnant une propriété oscillatoire.

Avec l'évolution du « sans-fil », les besoins en précision de fréquence se sont accrus, ce qui a conduit à l'évolution des recherches sur les oscillateurs à cristal.

Malgré cela, les oscillateurs à cristal sont limités, car la gamme de fréquences qu'ils peuvent fournir n'est pas assez étendue. Pour pallier à cette limitation, certaines méthodes proposent d'utiliser des capacités variables ou des circuits répondant aux harmoniques de la fréquence fondamentale du cristal, mais n'entraînent qu'une variation très légère de cette dernière. D'autres solutions ont aussi été longtemps utilisées. Notamment, celles qui consistent en l'usage d'une structure hétérodyne ou de mixage afin d'utiliser la somme et la différence entre deux fréquences d'oscillateurs. Elles ont permis d'avoir une plus large gamme de fréquences, tout en maintenant la stabilité de l'horloge des oscillateurs à cristal.

L'utilisation des boucles de verrouillage de phase (PLLs) est apparue depuis plusieurs années, présentés par Gupta [21] et Lindsey [25], et continue toujours d'être populaire. Plusieurs ouvrages [3],[5],[7],[10],[17],[26],[27],[32],[37],[39],[42], ont été publiés sur la théorie et la conception des PLLs dont l'utilisation est avantageuse, grâce à leur structure de boucle fermée à asservissement. Ils servent notamment à effectuer des recouvrements de porteuses, des recouvrements d'horloge, la démodulation de fréquence et de phase, la modulation de phase, la synchronisation d'horloges et la synthèse d'horloge.

La méthode de verrouillage de phase est donc la méthode de synthèse de fréquence qui est la plus communément utilisée pour la production des oscillations à hautes fréquences dans les équipements modernes de communication. Sa structure la plus simple est représentée à la **Figure** 2-1.

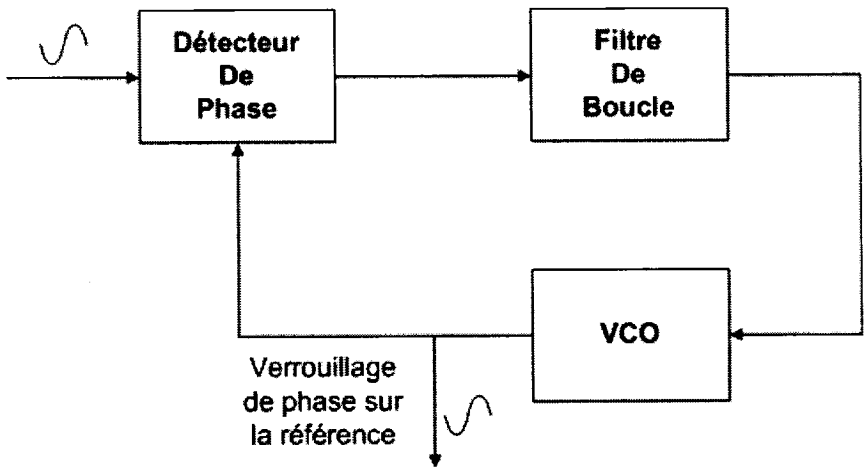

Figure 2-1 : Diagramme bloc général d'un PLL [D. Abravomitch, 2002].

Les éléments qui y sont illustrés font normalement partie de toutes les variantes de PLL, notamment :

- Le Détecteur de phase (PD). C'est un circuit non linéaire qui détecte la différence de phase entre les deux signaux oscillatoires et produit un niveau de tension quasiment continue et proportionnel à cette différence;

- Le VCO produit une fréquence de sortie qui dépend du niveau de tension de son signal en entrée;

- Le filtre de boucle (LF) qui peut être omis, est un filtre passe bas, qui sert à la conception de PLL de premier ordre. Il sert à choisir la composante continue venant du détecteur de phase;

- La connexion en boucle permet d'assurer le verrouillage de la fréquence.

La propriété qui permet de varier la fréquence est réalisée par l'Oscillateur Contrôlé par tension (VCO). L'avantage de ce circuit vient du fait que sa fréquence est contrôlable

électriquement, ce qui n'était pas le cas pour les anciens oscillateurs à fréquence variable (VFO), pour lesquels les propriétés du circuit devaient être modifiées pour réaliser le changement de fréquence. Les VCOs sont souvent structurés comme une boucle d'inverseurs. Ils sont parfois structurés comme des oscillateurs à relaxation ou des oscillateurs résonants. L'oscillateur en boucle est très souvent utilisé dans les topologies monolithiques, comme illustré à la Figure 2-3. Il se compose souvent d'un nombre impair d'inverseurs connectés les uns aux autres et formant une boucle grâce à une connexion en rétroaction. L'oscillateur à relaxation utilise souvent un Trigger de Schmitt pour générer une onde carrée stable.

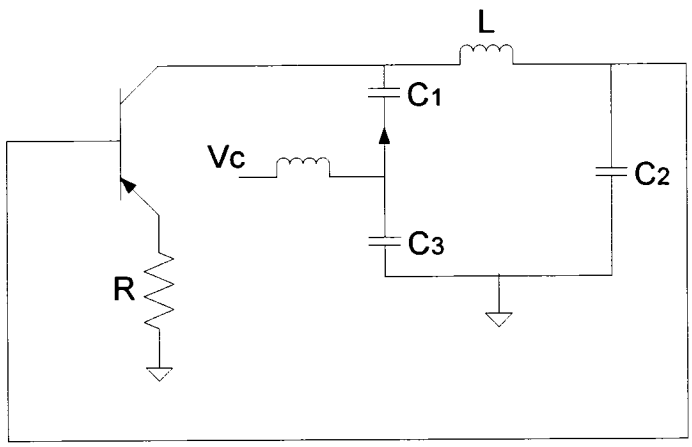

Figure 2-2 : Schéma de principe de circuit oscillant.

En ce qui concerne les VCO à oscillateurs résonnants, un circuit résonnant comme celui de la Figure 2-2 est placé sur le feedback positif à la sortie d'un amplificateur de tension à courant ayant un gain proche de l'unité. Cette approche est illustrée à la Figure 2-4. La fréquence est contrôlée grâce à la présence d'une diode polarisée en inverse. Ainsi, la capacité de jonction varie selon la valeur instantanée de la tension. Une telle diode à capacité variable est parfois appelée Varicaps™.

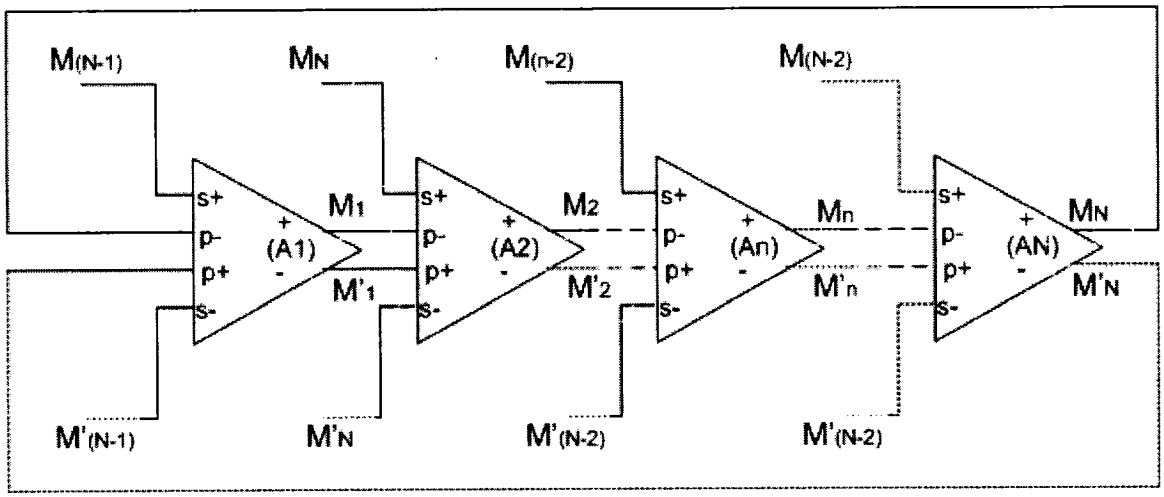

Figure 2-3. Oscillateur à boucle d'inverseurs [Yalcin Elper Eken et Al, 2004].

D'autres formes de VCO utilisées sont les cristaux contrôlés par tension (VCXO), et les oscillateurs de types YIG qui se différencient tout simplement des autres par l'architecture des circuits résonnants.

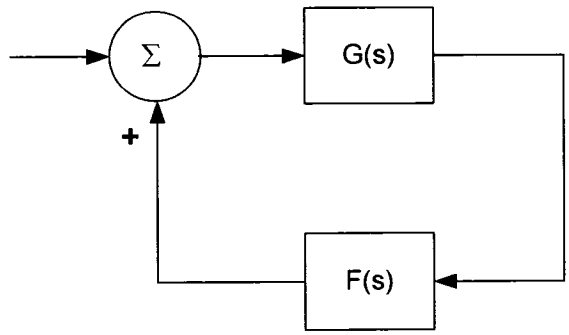

Figure 2-4 : Principe d'asservissement.

La conception présentée précédemment d'un PLL trouve ses limites avec l'évolution des circuits intégrés à grande échelle, étant donné son caractère analogique. Il est difficilement intégrable sur une puce à cause des éléments analogiques passifs comme les inductances et les capacités. Les PLLs numériques ont non seulement pour rôle de pallier à cet inconvénient, mais ils se servent aussi du fait que l'horloge présente dans

les systèmes numériques, les ordinateurs et les systèmes de transmission numérique, utilise des ondes carrées au lieu des sinusoïdes.

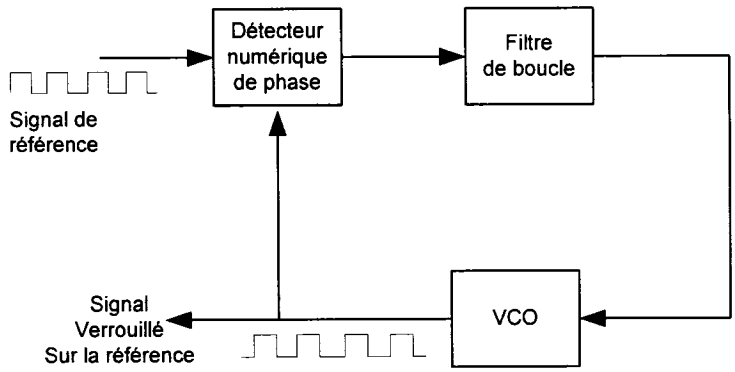

Figure 2-5 : Diagramme Bloc d'un PLL numérique classique.

La littérature [3] distingue deux sortes de PLLs numériques : les PLLs numériques classiques (CDPLL) dont l'architecture est illustré à la Figure 2-5, où seul le détecteur de phases est numérique, et les PLLs complètement numériques (ADPLL) qui utilisent des oscillateurs numériquement contrôlables (DCO) à la place des VCO, comme l'illustre la Figure 2-6.

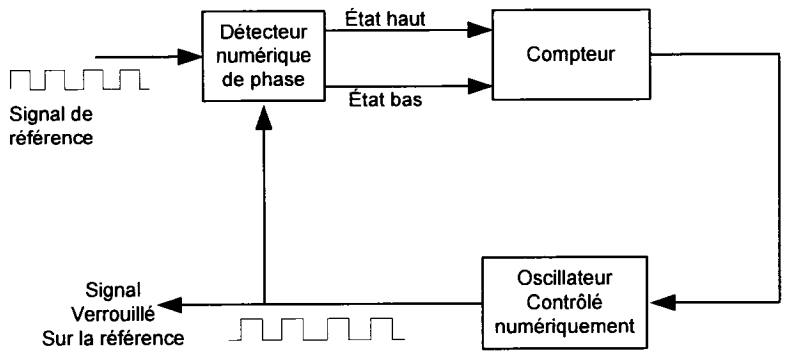

Figure 2-6 : Digramme Bloc d'un PLL complètement numérique.

La description précédente des PLL montre qu'ils permettent de générer un signal d'horloge stable et flexible dans le VCO, à la fréquence de référence, qui est celle de

l'horloge du cristal utilisé. Par contre, si l'on insère un diviseur de fréquence entre le détecteur de phase et le VCO, il sera possible de verrouiller le détecteur de phase à une fréquence qui est une fraction de la fréquence désirée dans le VCO. Par exemple, pour un diviseur par 2, et un cristal fonctionnant à 10 MHz, le détecteur de phase essayera de verrouiller le VCO à 20MHz. De manière générale, la fréquence du signal à la sortie du VCO est tirée de la formule suivante :

$$\frac{f_{VCO}}{N} = f_{Clk} \tag{2.1}$$

Où N représente le facteur de division, et f_{Clk} la fréquence de l'horloge de référence.

Une architecture plus complète du PLL est présentée à la Figure 2-7. Elle est à la base de nombreux travaux sur les circuits à fréquence variable, étant donné la possibilité de programmer le facteur de division.

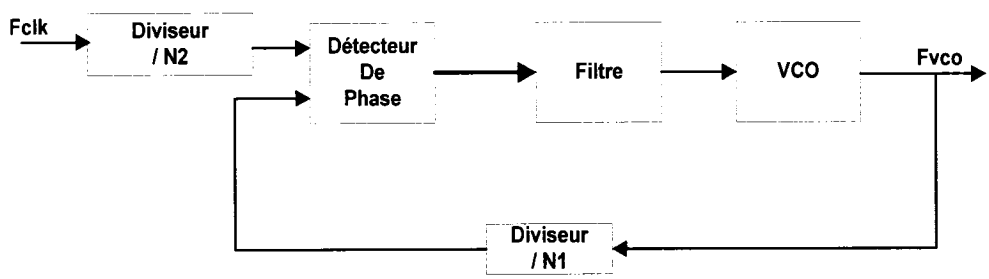

Figure 2-7 : Diagramme bloc d'un PLL.

Notons la présence d'un diviseur additionnel inséré entre le générateur de l'horloge de référence et le détecteur de phase, afin d'obtenir un rapport de division de la fréquence de l'horloge à la sortie du VCO, exprimé de la sorte :

$$f_{VCO} = \frac{N1}{N2} \times f_{Clk} \tag{2.2}$$

Le rapport N1/N2 est utile pour la synchronisation entre les signaux audio et vidéo, étant donné qu'ils sont de fréquences différentes. Le Tableau 2-1 nous montre un exemple des normes de la télévision numérique, illustrant les rapports possibles entre les fréquences audio et vidéo.

Tableau 2-1 Rapports de division communs entre fréquences vidéo et audio

		Fréquences Audio		
		4.0960 MHz	**5.6448 MHz**	**6.1440 MHz**
Fréquences Vidéo	**27.00000 MHz**	512/3375	392/1875	256/1125
	28.63636 MHz	5632/39375	616/3125	2816/13125
	35.46895 MHz	81920/709379	112896/709379	122880/709379
	36.00000 MHz	128/1125	98/625	64/375

Les termes respectifs des rapports de division indiquent les nombres de cycles d'horloge requis de chacun des signaux avant de se synchroniser entre eux..

Comme il a été précisé dans les paragraphes précédents, les facteurs de division peuvent être changés automatiquement selon plusieurs méthodes. Certaines architectures utilisent des diviseurs d'horloge dynamiquement programmables [7], [9], qui permettent de diviser la fréquence de l'horloge autant par des nombres pairs que par des nombres impairs.

Il existe d'autres formes d'architectures à PLL qui permettent d'effectuer une division fractionnaire de la fréquence de l'horloge en moyenne. Ce sont les synthétiseurs

fractionnaires dont un exemple est illustré à la

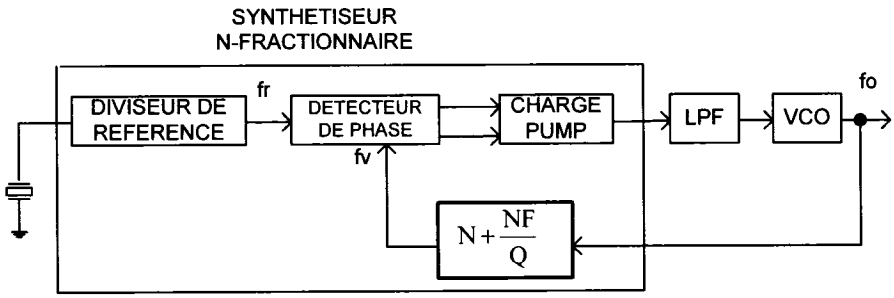

Figure 2-8.

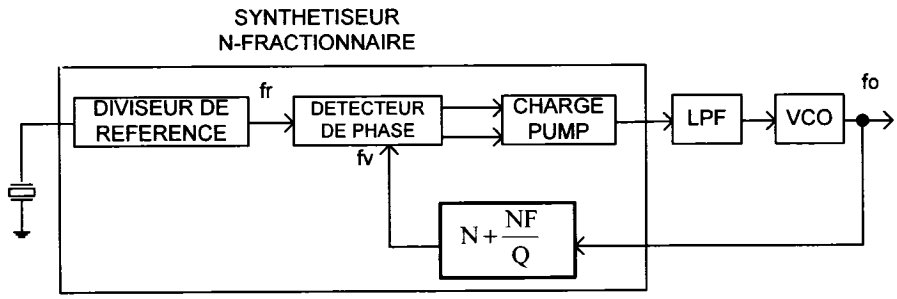

Figure 2-8 : Synthétiseur de fréquence N-fractionaire [Rhee W , 00].

La division fractionnaire y est interpolée en utilisant un compteur bi modulaire qui est commandé soit par un accumulateur tel qu'illustré à la Figure 2-9 , soit par un modulateur delta sigma. En effet, le diviseur programmable a une valeur de division de N pour un certain nombre de périodes de l'horloge, et change à la valeur N+1 pour un autre nombre de cycles d'horloge. La valeur moyenne de la fréquence obtenue en sortie est une valeur contenant une partie fractionnaire qui dépend du signal de contrôle et de sa résolution en bits. Plusieurs types [16][34][35] de circuits de synthèse N-Fractionaire d'horloge, avec plusieurs modes de contrôle des diviseurs programmables, existent dans la littérature.

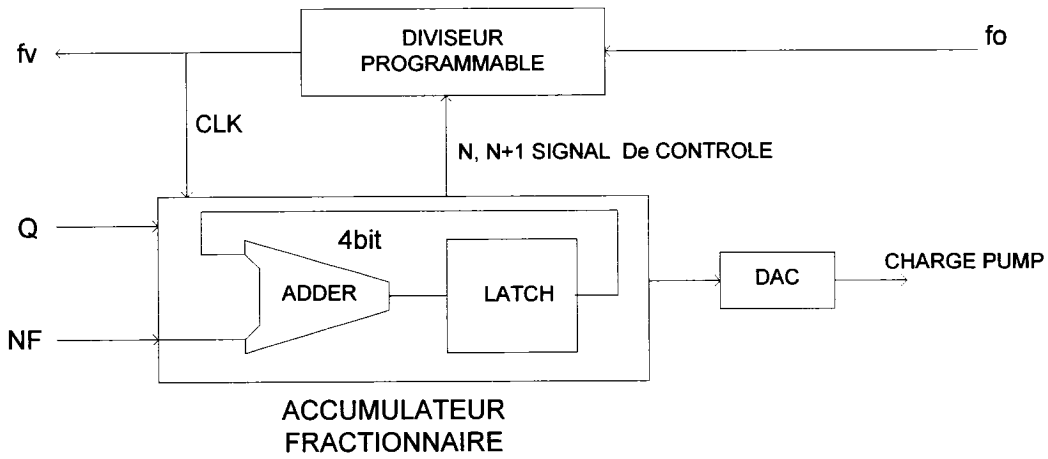

Figure 2-9 : Diviseur modulaire N-Fractionaire.

2.1.2 Synthèse numérique directe d'ondes (DDS)

La technique de synthèse numérique de fréquence (DDS) a vu le jour au début des années 1970, en tant que technique capable de générer des signaux sinusoïdaux avec une grande précision. La fréquence du signal à la sortie peut être contrôlée et ajustée à haute vitesse. Cependant ...L'article de J. Tierney [38] qui est l'un des pionniers de ce type circuit, présente un schéma global du DDS, explique sa fonctionnalité, et en énumère quelques applications possibles.

Dans le domaine des communications, le DDS est devenu une technique populaire compte tenu du besoin de produire des signaux sinusoïdaux précis, et de contrôler avec précision leur fréquence, phase et amplitude. Avec quelques modifications, cette technique est aussi utilisée pour générer d'autres types de forme d'ondes, tels les ondes triangulaires ou de forme arbitraire. D'autres applications incluent le domaine des radars, où la rapidité avec laquelle la fréquence change dans le DDS est un facteur

important. Les communications mobiles, telles que les systèmes de téléphonie cellulaire, utilisent le DDS pour générer des fréquences de référence.

Beaucoup d'articles dans la littérature présentent l'utilisation d'un circuit DDS dans chacun de ces domaines [19].

Ce circuit est aussi utilisé par D. Calbaza [12] pour effectuer la synchronisation entre les horloges vidéo et audio en DTV, car il permet de contrôler avec une précision accrue le rapport de division entre les fréquences de deux signaux – c'est à dire d'obtenir un signal qui a une fréquence contrôlée avec grande précision en utilisant une autre fréquence comme référence.

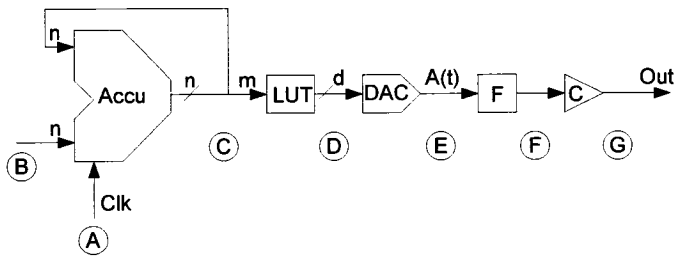

Figure 2-10 : Structure générale du DDS [Calbaza, 01].

La Figure 2-10 présente la structure générale d'un DDS. Le circuit DDS contient l'accumulateur de phase (Acc), une table de Conversion (LUT), un convertisseur numérique analogique (DAC), un filtre (F) et un comparateur (C).

L'accumulateur de phase ajoute l'incrément de phase (P) à la phase accumulée à chaque cycle d'horloge. Le nombre de bits de l'accumulateur, *n* dans ce cas, nous donne la précision du calcul. On observe que la phase accumulée, présente à la sortie de l'accumulateur, a une période répétitive déterminée par : le nombre de bits de l'accumulateur, l'incrément de phase et la période de l'horloge Clk [19].

La phase accumulée réduite de *n* bits à ses *m* bits les plus significatifs, est convertie par la table de conversion, LUT, dans la forme d'onde désirée. La plupart des circuits DDS

utilisent comme table de conversion une mémoire ROM, qui donne à la sortie la valeur numérique correspondant au sinus de l'argument représenté par les m bits les plus significatifs de la phase accumulée.

Cette valeur numérique est convertie en signal analogique par le convertisseur numérique analogique (DAC). Le filtre (F) est utilisé pour couper les harmoniques supérieures du signal produit par le DAC. À la sortie du filtre, on aura un signal sinusoïdal ayant une fréquence donnée par [15] :

$$f_a = \frac{N \times f_{Clk}}{2^n} \tag{2.3}$$

Où f_a est la fréquence de sortie et f_{Clk} est la fréquence de l'horloge Clk, et N est le nombre binaire présent à l'entrée de l'accumulateur de phase.

Le comparateur (C) permet de transformer le signal sinusoïdal en une onde carrée qui pourra être utilisée comme horloge pour les systèmes numériques. Ce signal est à la même fréquence que celle obtenue à l'équation 2.3.

À la sortie du DDS, la fréquence maximale qu'on puisse atteindre est la moitié de la fréquence de référence. En pratique, cette fréquence ne peut atteindre que le tiers de la fréquence de référence à cause des caractéristiques physiques du filtre.

Le signal à la sortie du DAC est de la forme suivante :

$$A_d(t) = \sin(\omega_a \cdot t) = \sin(2 \cdot \pi \cdot f_a \cdot t) \tag{2.4}$$

Le signal A (t) est représenté à la Figure 2-11 [6] et [18] :

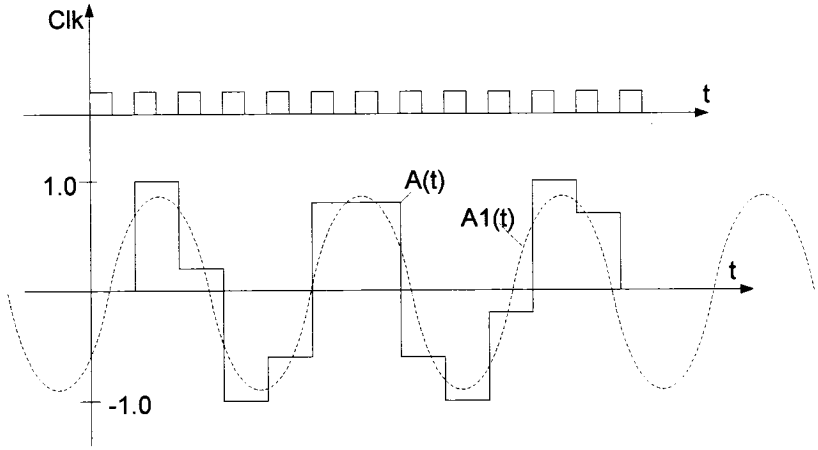

Figure 2-11 : Signal du DDS [Calbaza, 01].

A(t) n'est pas encore une sinusoïde, le filtre après le DAC permet d'obtenir la fondamentale A1(t) qui est un signal presque parfaitement sinusoïdal à la fréquence désirée. Son expression analytique est la suivante :

$$A_1 = \sin c(\frac{\omega_a \cdot T}{2}) \cdot \sin(\omega_a \cdot t - \frac{\omega_a \cdot T}{2}) = a_1 \cdot \sin[\omega_a \cdot (t - \phi_1)] \qquad (2.5)$$

On peut noter que cette composante est déphasée de la moitié de la période de l'horloge d'entrée Clk. Ce déphasage est exprimé par le terme $\phi_1 = T/2$. L'amplitude de cette composante est donnée par le terme :

$$a_1 = \sin c(\frac{\omega_a \cdot T}{2}) \qquad (2.6)$$

Étant donné le caractère analogique du filtre, les autres harmoniques du signal venant du DAC ne sont pas parfaitement filtrées, ce qui introduit une gigue.

Dans ce travail, nous ne ferons pas une étude de gigue, puisque le circuit que nous concevons est purement numérique, ce qui apporte un avantage face au DDS en termes de consommation de puissance et de plage de fonctionnement en fréquence.

2.1.3 Synthèse numérique directe de périodes (DDPS)

Calbaza [10] présente un nouveau type de circuit de synthèse d'horloge, le circuit de synthèse numérique de période (DDPS). Ce circuit produit une onde carrée, comparativement au DDS qui produit une onde sinusoïdale.

Ce faisant, la table de sinus LUT, le convertisseur numérique analogique et le filtre ne sont plus utilisés. Ce circuit a l'avantage d'apporter plus de simplicité, avec une réduction de la consommation de puissance grâce à l'élimination de la mémoire LUT. Il peut atteindre des fréquences en sortie plus élevées, et il s'en suit une réduction de la gigue de l'horloge produite.

Le diagramme bloc du DDPS est présenté à la Figure 2-12.

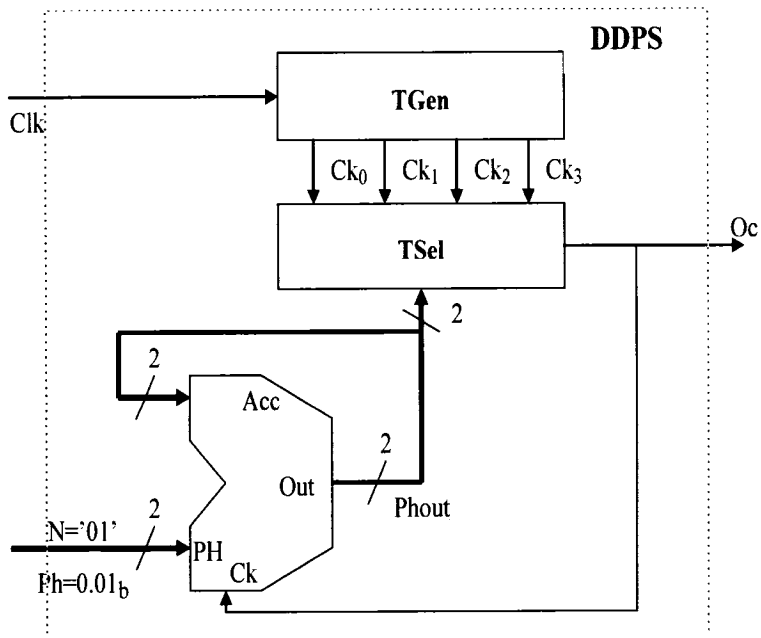

Figure 2-12 : Diagramme bloc du DDPS [Calbaza et Al, 99].

Le DDPS peut multiplier la fréquence de l'horloge de référence par un nombre fractionnaire. Le diagramme temporel de la Figure 2-13 illustre un exemple de son fonctionnement.

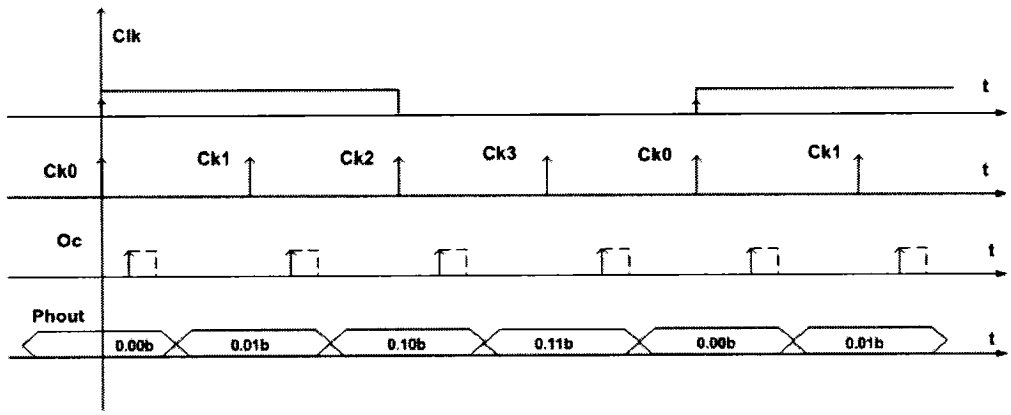

Figure 2-13 : Diagramme temporel du DDPS (Calbaza, 1999).

Le module générateur de transitions (TGen) produit plusieurs phases à la même fréquence que l'horloge de référence (Clk), à la même distance les unes des autres Le module de sélection des transitions (TSel) choisit les transitions du TGen. Cette dernière est propagée à l'horloge de sortie (Oc). Le choix de la transition est déterminé par l'accumulateur de phase (Acc).

Le fait de pouvoir sélectionner la transition qui pourra être propagée à l'horloge de sortie permet au DDPS de contrôler numériquement la période de sortie. Tandis que les circuits de la littérature couramment utilisés pour multiplier la fréquence propagent les transitions dans leur ordre naturel, produisant un signal de période fixe. Cette capacité est due au fait que le signal de sortie Oc contrôle directement l'accumulateur de phase. Cette astuce est aussi retrouvée dans l'article de Mair [29]. En utilisant le signal Oc, la

période peut être une fraction de la période (T) de l'horloge de référence Clk. Cette fraction est déterminée par l'incrément de phase (Ph).

Le diagramme temporel de la Figure 2-13 représente le cas où Ph=0.01b qui est la représentation binaire de ¼. À chaque coup d'horloge venant des impulsions de Oc sur l'accumulateur, les phases en sortie varient de 0.00b à 0.01b, 0.10b, 0.11b et la séquence se répète.

La séquence ci-dessus permet de sélectionner respectivement les front montants des phases de l'horloge : Ck0, Ck1, Ck2, Ck3 qui seront propagées l'une après l'autre comme impulsions de l'horloge de sortie Oc.

La fréquence à la sortie est déterminée par l'équation suivante :

$$f_{Oc} = \frac{f_{Clk}}{Ph} \tag{2.6}$$

Où Ph, inférieur à 1, est l'incrément de phase et f_{Clk} est la fréquence de référence de toutes les phases $Ck_{i..}$ Comme Ph est plus petit que 1, la fréquence du signal de sortie est plus élevée que la fréquence de l'horloge de référence.

Selon cette équation, on peut théoriquement produire une fréquence de sortie infiniment grande, mais évidemment, les propriétés physiques des éventuelles mises en œuvre vont limiter cette valeur.

En effet lorsque la séparation entre les phases est trop faible, l'impulsion à choisir pour obtenir la durée de période désirée arrive avant que les éléments du circuit qui constituent le chemin entre le TGen et le signal Oc ne prennent la nouvelle valeur dictée par l'accumulateur. Ceci est une cause d'aléas dans le cas où la mise à jour du chemin critique est très proche de la transition à sélectionner, ou ceci conduirait à sauter un cycle entier de la période de l'horloge Clk si cette mise à jour est faite trop tard, résultant à une division de fréquence exprimée comme suit :

$$f_{Oc} = \frac{f_{Clk}}{1.0 + Ph} \tag{2.7}$$

Choisir un Ph très bas n'est pas acceptable car la sortie résultante contiendrait des aléas. Pour déterminer l'intervalle dans lequel on pourrait utiliser n'importe quelle valeur de Ph, il faut tenir compte des variations de délais dues au procédé et les conditions d'opérations significatives (température, tension alimentation). La fréquence maximale du DDPS est donc déterminée par sa boucle critique : cette boucle inclut le délai entre la transition Oc, en passant par l'accumulateur qui permet de mettre à jour l'incrément de phase, auquel s'ajoute le délai de propagation d'un multiplexeur utilisé pour la sélection de la phase propagée en sortie. L'auteur de ce circuit estime qu'avec une implémentation optimisée, la fréquence du circuit tournerait autour du 1GHz avec la technologie CMOS 0,25µm, étude qui n'a pas été confirmée expérimentalement. Une étude sur la gigue a plutôt été réalisée, démontrant une amélioration face au circuit du DDS.

2.1.4 Autres méthodes de synthèse d'horloge

Plusieurs autres circuits qui sont des prédécesseurs ou des successeurs aux circuits présentés ci-dessus ont été rapportés. L'un des plus utilisé, très ressemblant au DDPS, est le synthétiseur de fréquence et de phase « flying-adder » [29]. Une illustration de cette architecture est présentée à la Figure 2-14. Les phases du VCO sont utilisées pour effectuer une multiplication de l'horloge. Un registre est utilisé à sa sortie pour obtenir un « duty cycle » de 50%.

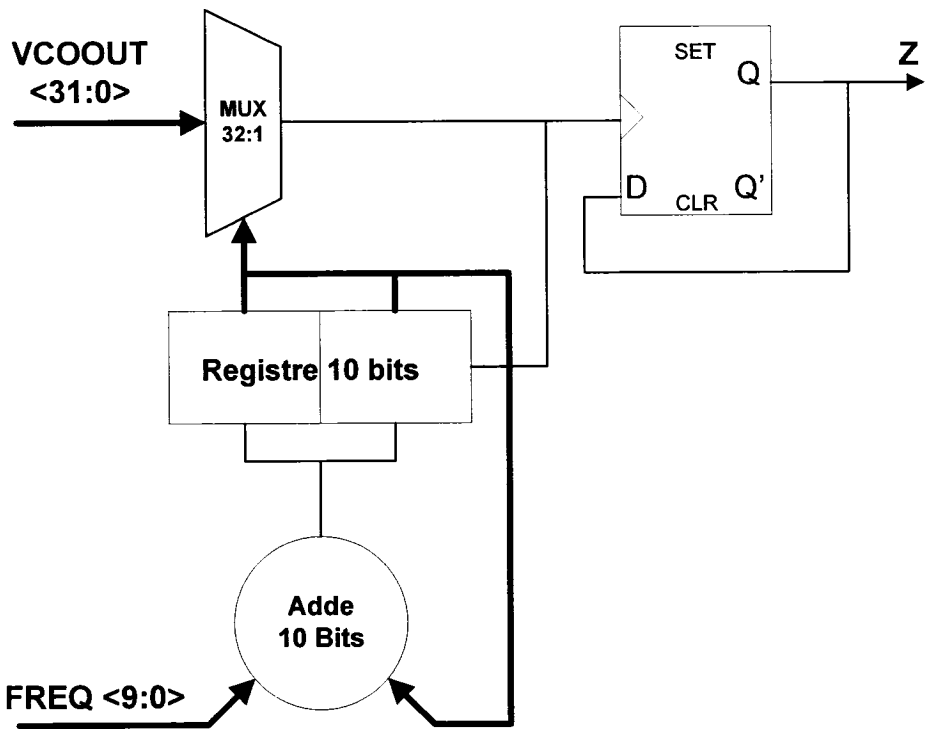

Figure 2-14 : Architecture d'un "Flying Adder".

2.2 Processeurs à faible consommation de puissance et à fréquence variable

Ce paragraphe présente une revue des techniques qui permettent une réduction effective de la consommation d'énergie dans les systèmes sur puces de génération récente en mettant une emphase sur les processeurs embarqués.

2.2.1 Modèle : puissance et énergie

La puissance dissipée dans les circuits VLSI en technologie CMOS est constituée de deux composantes relatives aux propriétés dynamiques et statiques de ces circuits :

$$P = P_{Dynamique} + P_{Statique} \tag{2.8}$$

Avec approximativement ;

$$P_{Dynamique} = CV_{DD}^2 f_{Clock} \tag{2.9}$$

Et

$$P_{Statique} = V_{DD} I_Q \tag{2.10}$$

Où C est la capacitance active du circuit durant son fonctionnement, f_{Clock} est la fréquence de l'horloge, V_{DD} la tension d'alimentation et I_Q le courant de fuite.

Cependant l'énergie qu'une pile ou une batterie fournirait à un tel circuit ne dépend pas de la fréquence d'opération. Sa composante dynamique est exprimée par :

$$E_{Dynamique} = \alpha \cdot CV_{DD}^2 \tag{2.11}$$

Avec α qui représente le coefficient d'activité du circuit.

2.2.2 Techniques de réduction d'énergie dans les processeurs embarqués

Pour une réduction effective de la consommation de l'énergie selon les équations du paragraphe précédent, plusieurs techniques s'appliquant aux processeurs embarqués ont été développées. Elles sont appliquées soit individuellement, soit combinées a une autre. Selon [4] et [28] ces différentes techniques peuvent être résumées comme suit:

- La technique de *voltage scaling* qui peut être appliquée dynamiquement (DVS) ou statiquement (CVS), nécessite plusieurs niveaux de voltage ;
- La technique de *clock gating* où le signal de l'horloge est contrôlé à l'aide de portes ET pour permettre d'arrêter le signal ;
- La gestion dynamique de puissance (DPM) ;
- La technique d'optimisation de puissance de la mémoire : méthode de partitionnement de l'accès à la cache ;
- Techniques d'optimisation de puissance par le contrôle du flux des données, par transformation du graphe de contrôle de flux de données (CDFG) ;
- Les méthodes d'optimisation de puissance au niveau d'abstraction des instructions ;
- La méthode de conception asynchrone globalement et synchrone localement (GALS) pour réduire la consommation de puissance du réseau de distribution de l'horloge

L'utilisation des ces méthodes de réduction d'énergie touchent tous les niveaux hiérarchiques de conception incluant le niveau algorithmique, architectural, le niveau circuit et la technologie de fabrication.

La conception de notre VSP est du domaine architectural avec une possibilité d'y appliquer les algorithmes d'ajustement dynamique du voltage (DVS) et de fréquence. Les lignes suivantes se consacreront aux différents travaux de la littérature concernant le DVS ainsi que la gestion dynamique de la puissance consommée.

2.2.3 Réduction d'énergie appliquée aux processeurs DVS

Historiquement, pour atteindre l'objectif de réduire la consommation de puissance dans les processeurs embarqués, les concepteurs ont longtemps utilisé la technique qui consiste à se mettre en mode de faible consommation lors des moments de repos. Or de plus en plus ces processeurs exécutent des tâches beaucoup plus sophistiquées qui requièrent un plus haut niveau de performance, ainsi les nouvelles applications audio et vidéo ainsi que les jeux vidéo s'exécutent pendant des périodes de temps énormes, ce qui entraîne une augmentation du rapport du temps d'exécution sur le temps de repos. Ceci étant, ces techniques de gestion de puissance, très efficaces lors des périodes de repos, ne permettent pas de préserver l'autonomie des batteries pendant les moments d'opérations actives.

Bien qu'on observe une évolution dans la conception des batteries (qui se traduit par une durée de vie prolongée et une réduction de leur taille), les demandes des nouveaux design de nouvelle génération en consommation d'énergie ont en revanche augmentées plus rapidement, les moyens conventionnels de gestion de puissance ne conservant pas l'autonomie des batteries à un niveau acceptable par les usagers.

Pour remplir les objectifs de performance et ceux de faible consommation de puissance, il suffit de permettre aux processeurs de rouler à différents niveaux de performances selon la charge de travail. Par exemple, un lecteur de vidéo MPEG demande plus de performance qu'un lecteur audio MP3. Il serait donc possible de ralentir le processeur

pour les applications MP3 sans pour autant perdre la qualité. Avec l'équation 2.11, l'énergie peut être sauvée en réduisant la tension d'alimentation du processeur pendant que sa fréquence est réduite.

La méthode DVS exploite le fait que la fréquence de pic du processeur implémentée en technologie CMOS est proportionnelle à sa tension d'alimentation. La Figure 2-15 illustre bien cette relation entre la fréquence et la tension d'alimentation en technologie CMOS 0,18µm.

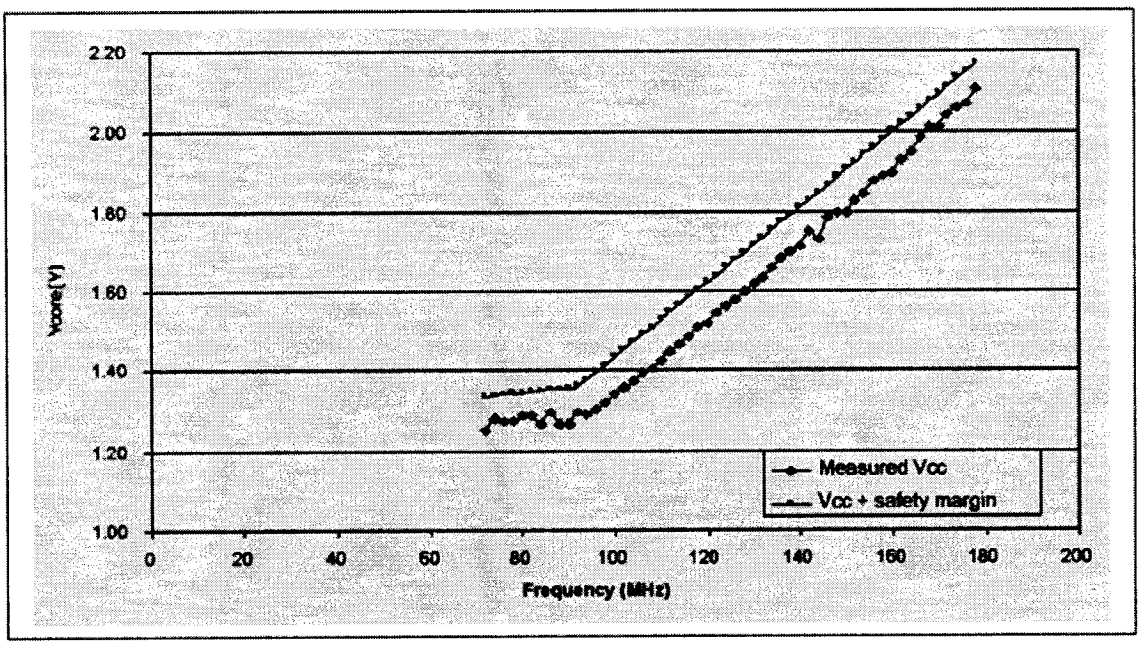

Figure 2-15 : Mesures voltage vs fréquence pour le ARM926EJ-S™ [Clive Watts, 2003].

Autour de la fréquence 90MHz on constate que l'intervalle des niveaux de tension a une limite qui ne peut être dépassée pour l'ajustement de tension.

La réduction du voltage et de la fréquence permet une réduction quadratique de l'énergie dynamique avec pour autre conséquence le rallongement du temps de fonctionnement. Étant donné que l'énergie conservée dans la batterie n'est pas infinie, cette méthode est très utile pour augmenter l'autonomie de la batterie.

La conception d'un processeur compatible DVS nécessite la présence non seulement d'un processeur à faible puissance supportant une gamme de niveaux de voltage et de fréquences, mais aussi un régulateur dynamique de voltage et de fréquence, le tout commandé par un gestionnaire de voltage et d'énergie. Le DVS est utilisée déjà dans certains processeurs commerciaux à faible puissance dont les caractéristiques sont illustrées au tableau 2-2.

Tableau 2-2 Caractéristiques de quelques processeurs DVS commerciaux connus

	Intervalle de Voltage	Intervalle de fréquences
IBM PowerPC 405LP [22]	1.0V – 1.8V	153MHz – 333 MHz
Transmeta Crusoe TM5800 [39]	0.8V – 1.3V	300MHz – 1GHz
Intel Xscale 80200 [23]	0.85V – 1.55V	333MHz – 733MHz

Kuroda [24] est l'un des premiers à avoir publié les résultats expérimentaux de la méthode de « *voltage scaling* » sur un processeur à utilisation générale. La tension minimale requise dans le processeur RISC R3900 pour la fréquence d'opération est toujours obtenue dynamiquement en comparant la fréquence avec le délai d'une copie du chemin critique du système alimentée à la tension minimale VDDL; ainsi le processeur est toujours alimenté au voltage minimum nécessaire. Le R3900 fonctionne entre 1.9V à 40MHz et 1.3V à 10MHz supportant toutes les fréquences intermédiaires.

Son régulateur de tension, à la Figure 2-16, est adaptatif à la fréquence de fonctionnement et consomme 140mW à 40MHz. [20] présente la plateforme Itsy, une expérience sur une plate forme Linux. Elle utilise le processeur StrongARM SA1100 qui supporte le *voltage scaling*. Seulement deux niveaux de tension y sont implémentés, 1,5V pour des fréquences au dessus de 162MHz et 1,23V en dessous, les réductions en énergie ne sont donc pas importantes. La différence de consommation de puissance est de 15% entre les 2 niveaux de tension. De meilleurs résultats ont été obtenus avec la plateforme SmartBadge [36] similaire à Itsy. Des mesures expérimentales de puissance en utilisant les opérations en temps réel de décodage audio MP3 et de décodage vidéo MPEG montrent qu'il est possible d'atteindre des gains en énergie de 40% avec cette plateforme.

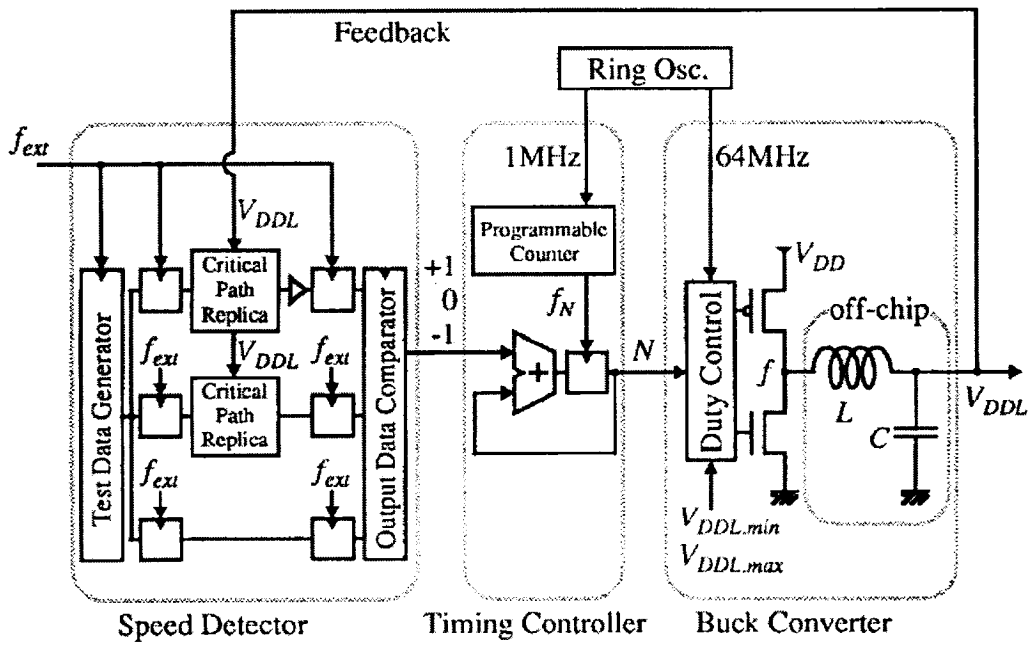

Figure 2-16 : Régulateur de voltage [Kuroda T, 24].

Burd T. [10] implémente un processeur basé sur un noyau de processeur ARM8 qui supporte l'échelonnage de la tension d'alimentation. Le circuit est fabriqué en technologie 600nm et contribue à l'économie de l'énergie. En mode haute performance,

il exécute à 80MHz et consomme 476mW à 3.8V. Avec une vitesse de 5MHz, il consomme 3,24mW à 1.2V. On note une réduction de la puissance par un facteur de 147 pour une performance réduite au seizième.

Le circuit de la Figure 2-17 qui se charge de la régulation de la fréquence, est une boucle fermée qui régule la tension d'alimentation en fonction de la fréquence. Il est implémenté sur deux puces distinctes : celle du régulateur et celle du CPU.

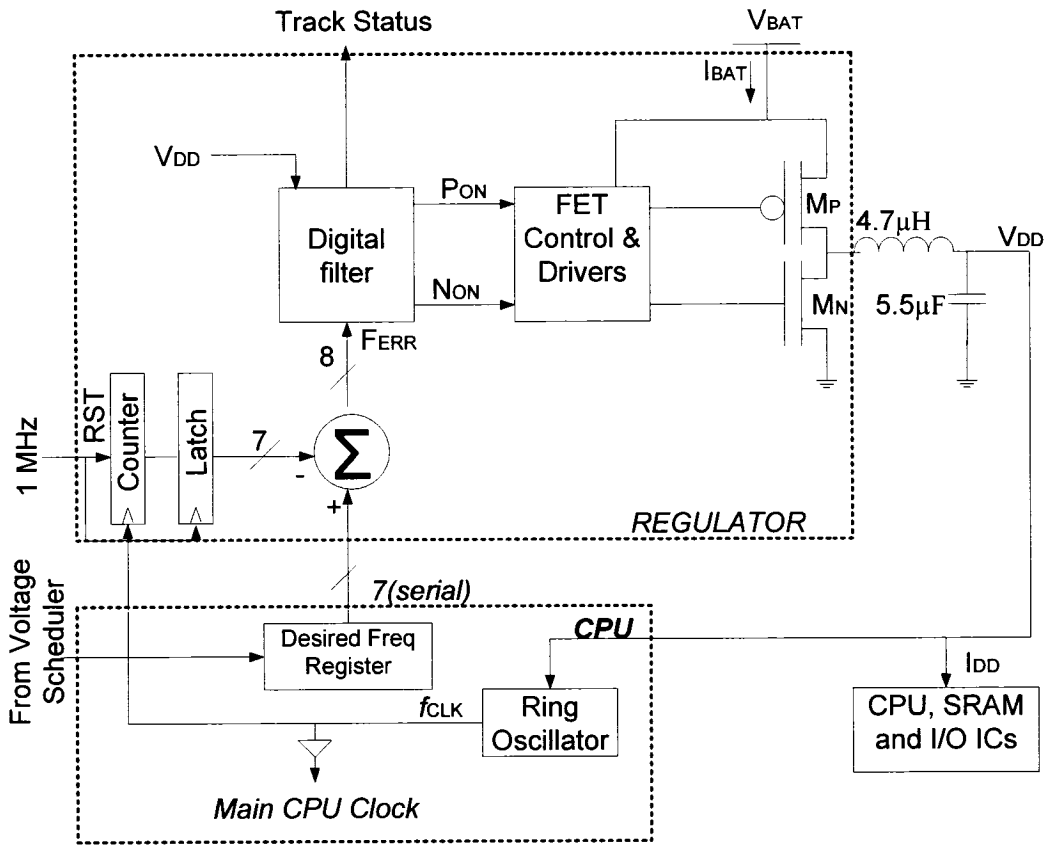

Figure 2-17 : Régulation par boucle de fréquence à voltage [Burd T.D, 10].

L'oscillateur en boucle du CPU fournit une horloge dont la fréquence est fonction de la tension d'alimentation V_{DD}. Le signal de l'horloge est envoyé sur la puce du régulateur et commande un compteur qui est réinitialisé à une cadence de 1MHz et permet de quantifier la fréquence en un mot de 7 bits. Cette valeur est soustraite de la valeur de fréquence désirée (en MHz) qui est donnée par le système d'exploitation, ce qui génère un mot de 8 bits représentant l'erreur en fréquence F_{ERR}. Le filtre numérique effectue une modulation de largeur et de fréquence d'impulsions (PMW/PFM) pour activer les transistors de puissance à effet de champ (TEC) via les signaux P_{ON} et N_{ON}. Ensuite le circuit de nivelage de tension convertit la tension de la pile VBATT (3.3-6.0V) à la tension régulée V_{DD} qui alimente le CPU pour fermer la boucle. La clé de l'adaptation aux changements de la température et du procédé est l'implémentation sur puce de l'oscillateur en boucle contrôlé par voltage. Il s'adapte aux conditions réelles de fonctionnement. Le temps de transition maximal lors du changement de voltage de 1,2V à 3,8V est de 70µs, avec la fréquence qui change de 5MHz à 80MHZ. Il existe une énergie dissipée lors de la transition qui est de 70µJ.

L'article [30] présente la conception d'une plateforme autour d'un processeur 32-bits basé sur le PowerPC, qui supporte en plus les algorithmes DVS. Il permet aussi d'ajuster la fréquence de l'horloge en temps réel, s'adaptant dynamiquement aux changements des demandes en performance aux contraintes de faible consommation de puissance. À 1,8V il atteint une fréquence de 380MHz pour une consommation de 500mW, et réduit sa consommation à 53mW pour un voltage de 1,0V et une fréquence de 152MHz. L'ajustement de la tension et de la fréquence peut aussi être effectuée par logiciel, ainsi [32] propose des algorithmes d'ordonnancements selon la priorité en énergie, qui utilise les descriptions des charges de travail des applications pour gérer l'exécution des tâches. Le design est basé sur le processeur StrongARM d'Intel et supporte les applications à puissance consommée réduite. La plateforme supporte 128 niveaux de tension. À 0,79V il roule à 59MHz et à 1,65V la fréquence est de 251MHz.

CHAPITRE 3
Méthodologie d'implémentation

La méthodologie de conception du processeur à vitesse variable est présentée dans ce chapitre sous deux volets :

- la méthodologie en conception ASIC pour le synthétiseur d'horloge à période variable, le VPCS ;
- la méthodologie de conception pour FPGA du processeur à vitesse variable.

Ces méthodologies suivent les mêmes principes que celles de base mais avec un accent sur les différences qui s'appliquent à la méthode de cycles d'horloge variables.

3.1 Méthodologie de conception ASIC du VPCS

Le flot de conception du VPCS illustré à la Figure 3-1 est le même que pour tout circuit implémenté en technologie ASIC. En premier lieu, il faut définir les spécifications du circuit à réaliser selon les objectifs à atteindre. Ensuite il faut passer à la description en langage de haut niveau (VHDL) et sa compilation, qui formera la description RTL du design et permettra de faire la simulation fonctionnelle au niveau RTL. Une fois la fonctionnalité du circuit vérifiée après simulation RTL, on procède à la synthèse du circuit. Durant cette étape nous nous concentrons sur l'analyse du timing et l'évaluation de la puissance dynamique du circuit pour optimiser le design. Nous nous sommes arrêtés à la simulation après synthèse, autrement dit la simulation au niveau des portes logiques. La validation des résultats en ce qui concerne les performances en fréquence du circuit est effectuée durant cette dernière étape. Les sections suivantes décrivent en grandes lignes, l'implémentation de notre générateur d'horloge.

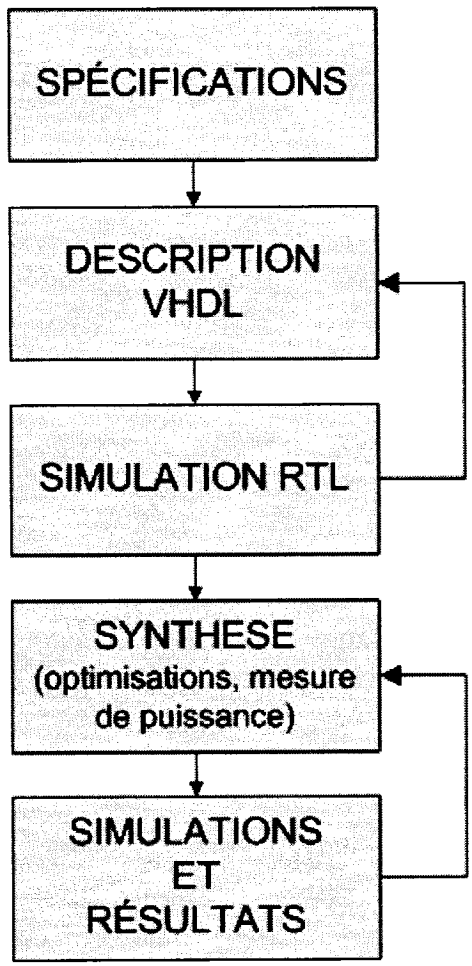

Figure 3-1 : Flot de design pour implémentation en ASIC.

3.1.1 Spécifications

Ce paragraphe décrit les spécifications du VPCS qui ont mené à la concrétisation de sa réalisation.

Le cahier de charge initial voulait qu'on puisse obtenir un circuit qui change la période d'une horloge selon un mot de commande en entrée. Le défi était ensuite d'effectuer le changement sans délais, puis de pouvoir le faire à chaque cycle selon la commande

présente. Ainsi, à la transition montante marquant la fin du cycle précédent, le calcul de la période suivante doit commencer.

Inspiré des récents DLL et des travaux sur le DDPS, il s'agissait de pouvoir obtenir à la sortie un signal de fréquence plus élevée que celui en entrée en sélectionnant les phases intermédiaires de l'horloge de référence.

La description du circuit est faite en VHDL pour lui donner un caractère purement numérique, le calcul de la durée de la période à synthétiser est fait par une Machine à états (FSM), la description du circuit est générique c'est-à-dire qu'on peut configurer le circuit selon le nombre de phases disponibles en autant que ce nombre est une puissance de 2. Le diagramme du flot de donnée de la FSM se trouve en Annexe B. Le mot en entrée qui est le facteur multiplicatif est un nombre arbitraire fractionnel à point fixe, la partie entière étant large de m bits, la partie fractionnaire de $\log_2(n)$ bits avec n qui représente le nombre de phases disponibles en entrée.

Le circuit sera utilisé dans un processeur supportant autant les applications de haute performance que celles à faible consommation d'énergie. Il s'agit donc de pouvoir générer différentes fréquences selon ces applications.

3.1.2 Description VHDL et simulation

La description VHDL selon les spécifications précédentes est présentée en Annexe A. L'analyse et la simulation du circuit sont faites grâce au simulateur HDL *Modelsim®*, pour compiler le programme en langage VHDL et faire une simulation fonctionnelle selon les vecteurs en entrée précisés par le banc de test. Il s'agit ici de vérifier qu'on obtient bien à la sortie la forme de signal escomptée. Les délais ne sont pas pris en compte, les phases de l'horloge sont fournies par le simulateur.

3.1.3 Synthèse et optimisations

À l'aide de l'outil *Design Analyzer*™ de *SYNOPSYS*®, la synthèse et l'analyse statique du Timing dans le circuit est effectuée. Après la compilation, le design est transformé en portes logiques prêt à être optimisé selon les contraintes et spécifications.

L'analyse du timing est différente des circuits conventionnels dans la mesure où la sortie du VPCS est réutilisée comme horloge pour certains registres. Il a fallu créer une horloge supplémentaire artificielle pour ce signal afin d'avoir des résultats plus réalistes, puisque l'analyseur ne comprend pas ce qu'implique la boucle dans le circuit. De plus, l'existence de plusieurs phases de l'horloge utilisées rend la tache difficile à l'outil d'analyse. Il était donc nécessaire de spécifier une valeur de fréquence au signal de sortie correspondant à la fonction voulue. (Par exemple, avec une fréquence de référence CLOCK_REF de 50MHz, pour multiplier cette fréquence par 4, on spécifie que le signal de sortie devrait être à 200MHz). Pour une version du circuit à 4 phases, l'analyse du timing nous montre l'existence de 5 domaines d'horloges et les distances entre les phases des différents domaines sont prises en considération dans le calcul du délai critique. À cette étape nous faisons une estimation de la fréquence grâce à la valeur de la période maximale admissible selon l'outil de Synopsys. Si la fréquence de l'horloge en sortie est trop élevée, il y a une violation des règles de contraintes de temps selon la technologie CMOS 180nm, il faut donc modifier cette fréquence jusqu'à ce qu'elles soient respectées. Une autre synthèse est nécessaire une fois toutes les horloges créées et les modifications des fréquences faites, afin d'obtenir un circuit qui respecte les contraintes.

3.1.4 Simulation après synthèse

L'outil de simulation est *Modelsim®*. Il permet d'effectuer une simulation assez fidèle du comportement du circuit. Pour une simulation plus proche de la réalité, on applique au circuit obtenu après synthèse, les délais de chaque élément qui se trouvent dans le fichier contenant ces délais, d'extension *sdf*. Le dit fichier étant créé automatiquement par le *design compiler®* de *Synopsys* selon les bibliothèques standards de la technologie CMOS 180nm. La commande utilisée pour créer le fichier des délais est « ***write – constraints nonmdufichier.sdf*** ». Le fichier ainsi obtenu est chargé dans Modelsim afin de tenir compte des délais des portes. Les formes d'ondes obtenues ne sont pas exemptes d'éventuels aléas et représentent le comportement du circuit sans tenir compte des capacités parasites ou de l'effet ajouté au cas où on implémenterait le circuit sur une puce. Tous les cas de simulations sont couverts (division et multiplication de la fréquence de l'horloge avec ou sans partie fractionnaire), il suffit de préciser les vecteurs d'entrée du VPCS correspondants à chacun des modes de fonctionnement.

3.1.5 Estimation de la puissance dynamique consommée

Le cheminement général de l'analyse de la consommation de puissance est illustré à la Figure 3-2. Pour des circuits à une seule phase d'horloge, l'estimation de la puissance se fait assez aisément car l'outil *Power Compiler* de *Synopsys* peut estimer fidèlement la puissance consommée par le circuit sans avoir à effectuer une simulation du circuit. Il suffit de préciser la fréquence désirée, et d'utiliser la commande ***report power –out fichierdesortie.out*** pour avoir le rapport contenant les détails de la puissance consommée. Il faut aussi annoter les changements d'états (switching activity) de tous les signaux du circuit grâce à la commande ***set_switching_activity*** à laquelle on spécifie les paramètres suivants de façon arbitraire selon les caractéristiques de l'horloge:

- la probabilité P_1 que le signal soit à 1 dans un cycle d'horloge ;
- le nombre de fois TR que le signal change d'état (0,1) à l'intérieur d'un cycle d'horloge.

Lorsque les changements d'états dans le circuit ne sont pas annotés, l'outil estime que le signal d'horloge défini a une probabilité par défaut $P_1 = 0.5$ et la fréquence de changements d'états par défaut est $TR = 0.5f_{CLK}$.

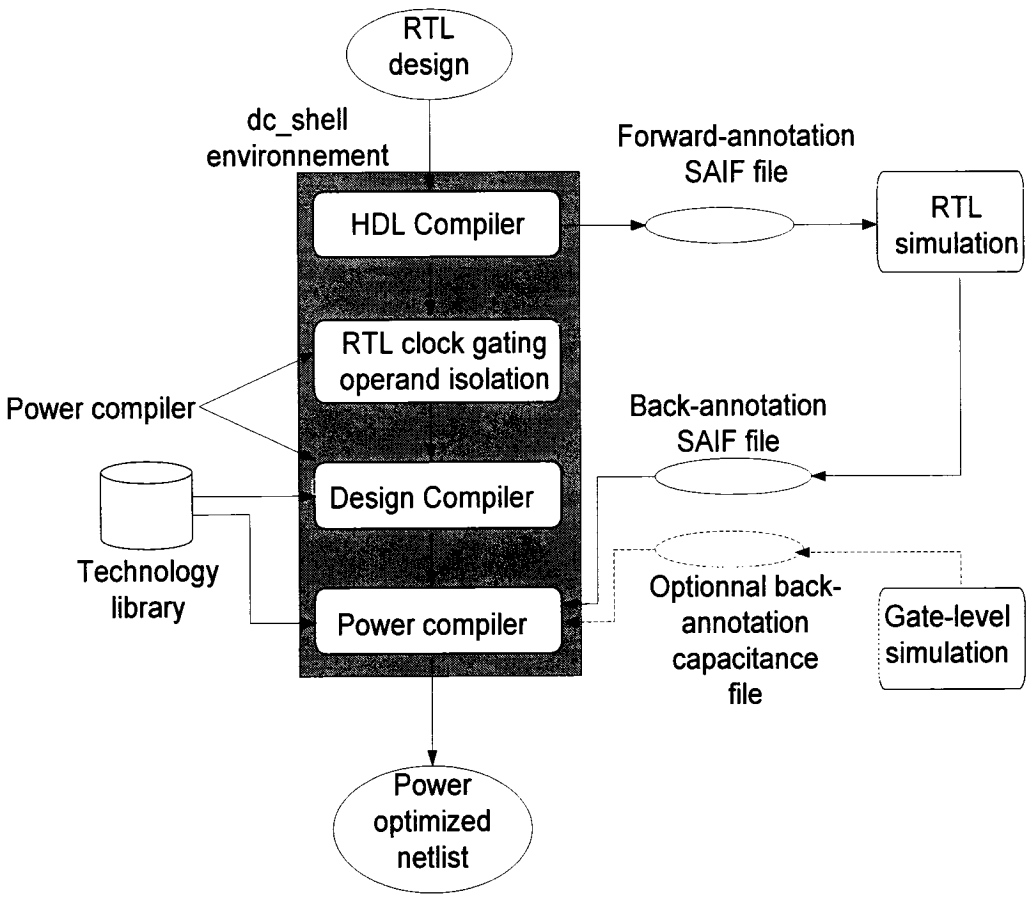

Figure 3-2 : Méthodologie et flux de donnée de l'analyse de puissance.

Avec le VPCS, il faut absolument annoter le circuit après simulation car les changements de phases durant le fonctionnement du circuit, ainsi que le fait que les signaux générés sont des impulsions, ne nous permettent pas d'annoter arbitrairement le

design, et encore moins d'utiliser les valeurs par défaut de l'outil d'analyse. Pour ce faire, il faut générer un fichier d'extension *saif* (Switching Activity Input File) tel qu'illustré à la Figure 3-3 qui contient toutes les informations sur les activités dynamiques du circuit après simulation (Backward SAIF) pour ainsi annoter le design à analyser.

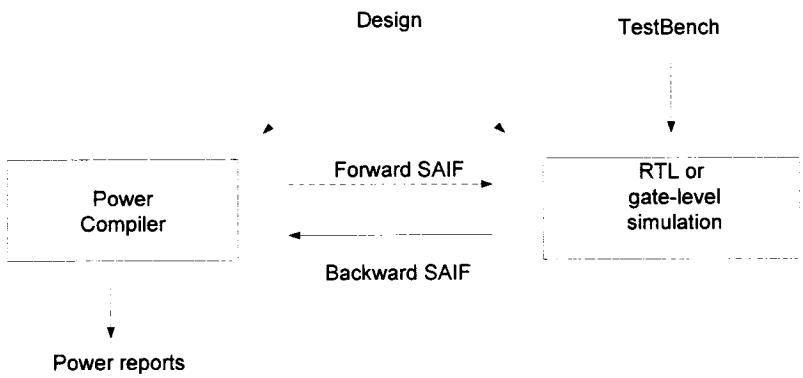

Figure 3-3 : Flot de création de fichiers saif.

Le fichier d'annotation *saif* est généré par le simulateur *Modelsim* en passant par la création de fichiers *vcd* (value change dump) qui contient les informations de l'activité du circuit après simulation sous un autre format. Ces derniers sont créés grâce à la commande *vcd add*, puis la conversion se fait grâce à la commande *vcd2saif*.

Une fois le fichier *saif* utilisé pour l'annotation du circuit, la puissance dynamique peut être estimée. Les outils nous montrent qu'elle est constituée en 2 parties. La puissance interne qui est celle que dissipe toute cellule à l'interne, et la puissance de transitions (switching) qui est celle dissipée par les capacités de charges en se chargeant et se déchargeant tour à tour.

3.2 Méthodologie d'implémentation FPGA pour le VSP

En somme, la méthodologie d'implémentation du VSP est similaire à celle utilisée pour toute implémentation de systèmes sur puce programmable (SOPC), plus précisément sur FPGA. L'outil fourni par Altera est Quartus II. Il assiste le concepteur depuis l'élaboration du design jusqu'au placement et routage du circuit suivant le flux illustré à la Figure 3-4.

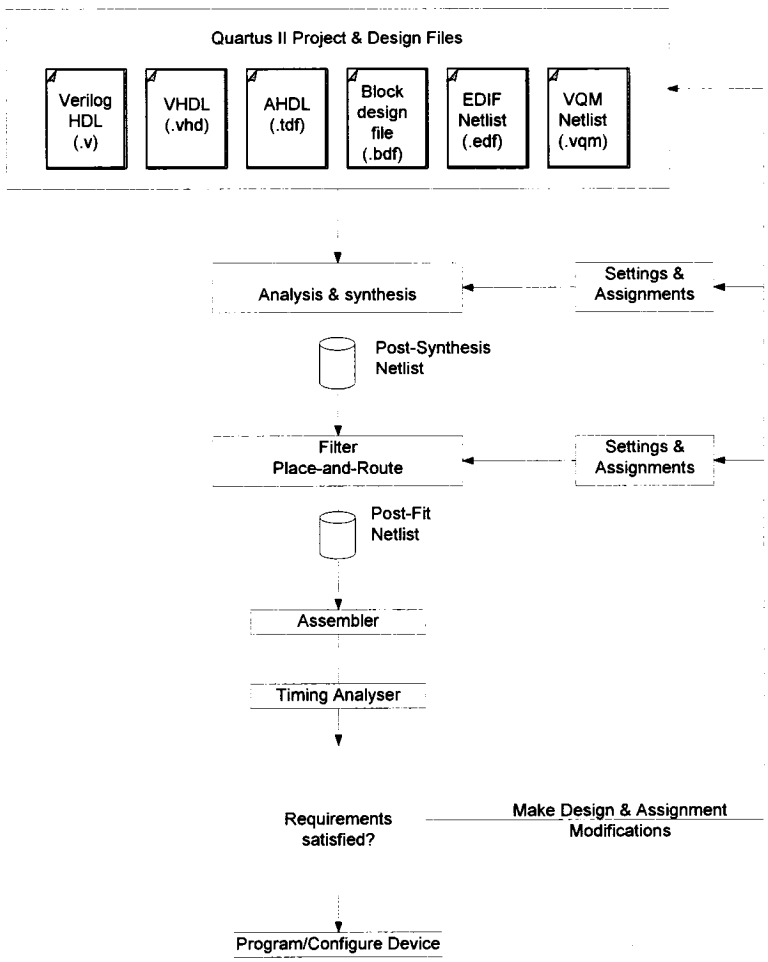

Figure 3-4 : Flot de design avec Quartus II [Altera design flow].

3.2.1 Spécifications et configuration du système

Les spécifications du VSP requièrent un processeur duquel on extrait les signaux de l'état du pipeline, qui seront utilisés comme commande du VPCS. Ceci requiert que la logique interne du processeur soit pleinement accessible. Une fois les opcodes venant du pipeline et de la mémoire d'instructions extraits, il faut s'assurer qu'à chaque cycle la durée de la période d'horloge est toujours plus longue que l'étage du pipeline le plus long, ceci en comparant les différents opcodes issus des différents étages.

Il peut arriver dans le pipeline des aléas provenant des données et même du contrôle, créant ainsi des cycles de suspensions. Aussi certaines instructions peuvent s'exécuter en plusieurs cycles, il est donc nécessaire de prédire à quels moments on aurait un cycle de plus, pour que la modification de la durée s'effectue au bon cycle. Ces événements sont signalés par les signaux *commit* et *pipe_run* issus du processeur.

Le Nios est un processeur RISC configurable contenant un pipeline à 5 étages. Le noyau logiciel est disponible, selon les choix du concepteur, sous les formats VHDL ou Verilog, et sa configuration initiale se fait grâce à l'outil SOPCBuilder disponible pour la plateforme de développement Stratix. Un exemple typique de système généré avec SOPCBuilder est illustré à la Figure 3-5.

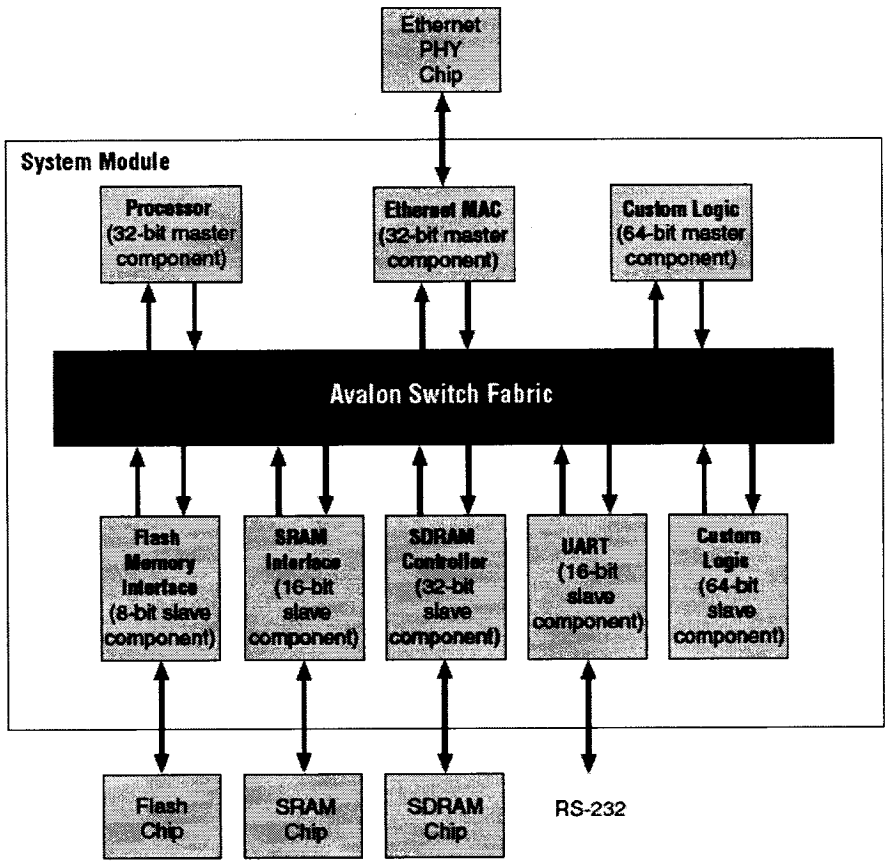

Figure 3-5 : Exemple de système généré par SOPCBuilder.

Il permet de configurer le système au complet en quelques minutes, en connectant plusieurs composants autour d'une architecture de bus. Il génère ainsi un bus appelé *Avalon® Switch fabric* qui gère toute la logique pour connecter les périphériques à l'intérieur et hors du FPGA Stratix. Une fois le système configuré, SOPCBuilder génère le code VHDL de tout le système. Compte tenu de la nécessité de modifier le code source du noyau du processeur, un défi à relever est de comprendre et analyser le code automatiquement généré afin de déterminer les registres du pipeline.

Le VSP devrait pouvoir supporter les algorithmes de DVS, donc pouvoir fonctionner à des voltages différents. Une instruction spécialisée est donc créée pour changer le mode

de fonctionnement. Globalement, cette instruction sera utilisée par le logiciel du système d'exploitation responsable de l'ordonnancement selon la technique de DVS.

3.2.2 Analyse du jeu d'instructions

Le jeu d'instruction du Nios contient un peu plus de 70 instructions. Le format est de 16 bits quel que soit la configuration choisie (16 ou 32 bits, qui correspondent à la largeur du bus de donnée). Il supporte les programmes compilés en C ou en C++, permettant d'effectuer les opérations standard arithmétiques et logiques ainsi que les opérations sur les bits, l'extraction des octets, les mouvements de données, la modification du contrôle et les instructions conditionnelles. Le Nios contient 32 registres, dont l'adressage est effectué par 5 bits du champ instructions. La plupart des instructions sont exécutées en un cycle, sauf pour le cas de la multiplication et des opérations de décalage, qui respectivement consomment trois et deux cycles.
L'instruction spécialisée *setmode* met à jour un module registre qui permet de choisir la bonne banque de délais selon que l'horloge doit être globalement ralentie ou pas. La valeur du registre est modifiée en 1 cycle. Le registre utilisé est le registre K qui contient généralement les valeurs du champ préfix. Le chargement de ce registre se fait donc en invoquant l'instruction *setmode* sans utiliser un registre de donnée général.

3.2.3 Profilage des délais d'instructions

La caractérisation du temps d'exécution des instructions d'un processeur à Architecture de Havard ne peut être effectuée de manière statique. En effet, les outils CAD ne peuvent pas distinguer les délais dans les portes logiques selon les différents vecteurs en entrée. Seul le délai du chemin critique peut être déterminé de façon statique avec les outils actuels. Pour distinguer les différents temps d'exécution critiques pour chaque instruction, il faut rouler une simulation contenant l'instruction à évaluer : le vecteur en

entrée devrait être celui qui nécessite l'utilisation du chemin critique pour l'opération à effectuer. Par exemple, l'addition dans une Unité Arithmétique Logique (ALU) de 1 au nombre 0xFFFF utilise la logique du Carry-in qui est le chemin critique pour cette opération.
Le processeur étant pipeliné, chaque étage est constitué d'un chemin critique selon les opérations à effectuer. Il s'agit donc d'estimer durant la simulation, le délai consommé par chaque étage. Cette étape de la conception est itérative car la seule façon d'estimer ce délai est :

- d'appliquer un vecteur de test significatif, qui est une valeur extrême de la donnée à manipuler par l'opération ;
- d'augmenter la fréquence de l'horloge jusqu'à obtenir celle pour laquelle le résultat escompté est corrompu, ainsi la fréquence maximale de fonctionnement de cette opération serait la dernière pour laquelle le résultat obtenu est juste.

Étant donné que la méthode choisie est d'allouer dans la mémoire contenant les informations sur les délais, une valeur par instructions, l'adressage de cette mémoire se fera sur 7 bits (capacité de 128). Il proviendra d'une uniformisation de la largeur en bits des champs des instructions, grâce à la logique additionnelle, étant donné que les opcodes ne sont pas tous de la même largeur.

3.2.4 Adaptation du VPCS en technologie FPGA

Le VPCS précédemment conçu pour une technologie ASIC est redéfini étant donné les différences de technologie avec le FPGA Stratix. En un premier lieu, on ne peut avoir que 4 phases équidistantes disponibles dans le Stratix, d'où la nécessité de n'utiliser que la version à 4 phases. Ensuite il y a une baisse de performance qui réduit la période minimale admissible de l'horloge à la sortie du VPCS, de 2ns à 4ns. La logique qui

permet de générer les impulsions fonctionne très bien en ASIC mais crée des problèmes au niveau de la distribution de l'horloge dans le FPGA. Cette logique est donc remplacée par un réseau de portes logiques ET/OU qui génère une horloge compatible, et un buffer est utilisé à la sortie du VPCS lors de la synthèse.

3.2.5 Estimation de la puissance consommée

Une fois le circuit synthétisé, le placement et le routage faits, le logiciel Quartus permet d'effectuer la simulation du circuit en estimant la puissance dynamique consommée durant l'intervalle de simulation. Le programme test est un programme en assembleur qui contient tous les types d'instructions (lentes et rapides, accès mémoire, calcul, transfert de données et branchement). Il contient une boucle et incrémente une variable à chaque passage de la boucle afin de conserver le nombre de boucles réalisées. Ce processus nous permettra d'évaluer la performance atteinte.
La simulation est faite sur un intervalle de temps de 100µs en visualisant les signaux du pipeline et les données à manipuler pour s'assurer du bon fonctionnement. L'intervalle pour l'estimation de la puissance est précisé par le concepteur, ainsi la période d'initialisation (Reset) a été écartée pour nos mesures. La puissance estimée par le logiciel est la puissance consommée par la totalité du Stratix y compris la puissance de « stand-by ».

3.2.6 Configuration et prototypage

La carte de développement qui sert de plateforme de prototypage est illustrée à la Figure 3-6. La carte est connectée au port parallèle d'un PC grâce au JTAG qui permet la configuration du FPGA.

Le port RS232 de la carte est connecté à un port série du PC pour réaliser la communication entre l'ordinateur et le prototype. La communication se fait à une cadence de 115 200 bit/s grâce à un UART dont l'horloge en entrée est passée dans un diviseur pour émettre les caractères à la cadence compatible à celle du port série. La valeur de ce diviseur est fixée lors de la configuration initiale, mais peut être aussi programmée. Cette dernière hypothèse n'est pas avantageuse pour notre méthode étant donné qu'il y a un délai de traitement avant de mettre à jour cette valeur.

Comme il est indiqué dans l'article du chapitre 4, pour éviter des problèmes de synchronisation entre les 2 domaines d'horloge, l'horloge du module maître du UART est connectée à l'horloge de référence initiale, et la communication se fait de façon asynchrone suivant un protocole de *handshaking* entre le maître et le bus Avalon. Ainsi la cadence fixe de communication est respectée entre la carte et le PC

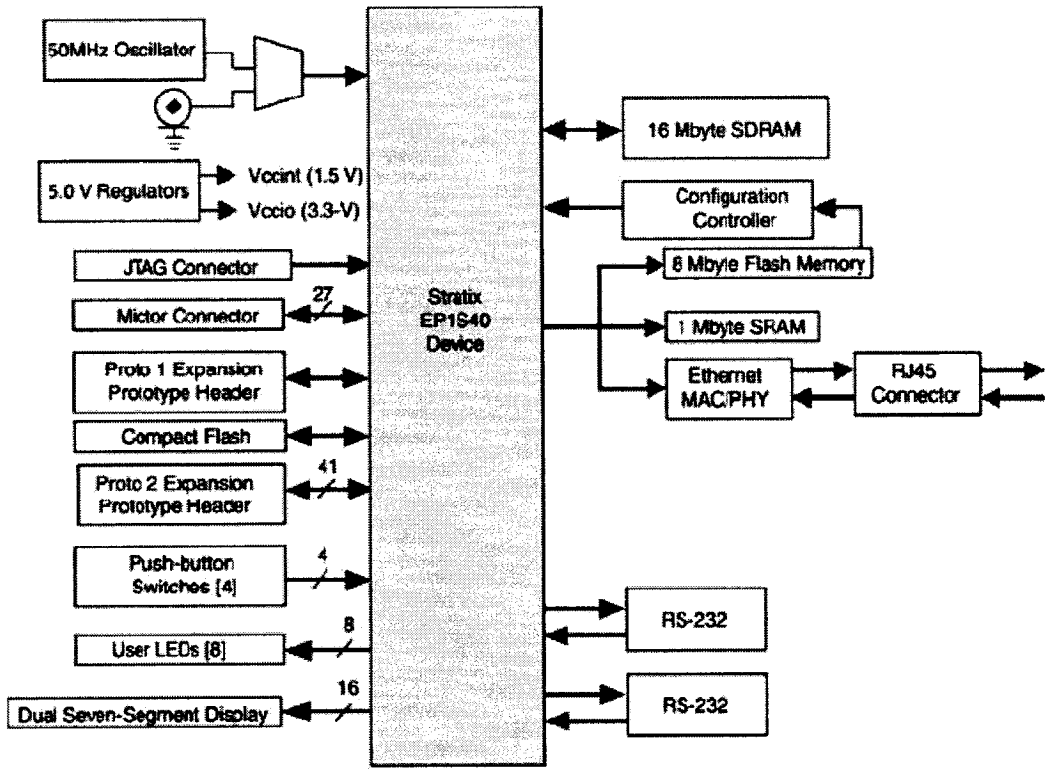

Figure 3-6 Diagramme Block de la carte de développement [Altera Corporation].

Le programme test décrémente d'une unité les nombres à partir de 99 et l'affiche sur l'afficheur 7-segments et en même temps à l'écran du PC connecté à la carte. Un affichage correct indique le bon fonctionnement du circuit.

CHAPITRE 4

Processeur embarqué à faible consommation de puissance avec vitesse variable à chaque cycle d'horloge

Le chapitre précédent montre la méthodologie utilisée pour atteindre l'objectif fixé. En résumé, le projet s'est réalisé en deux étapes :

- la conception du circuit de synthèse d'horloge à période variable (VPCS) ;
- la conception du processeur à vitesse variable (VSP).

Ce chapitre représente une intégration des travaux effectués et a fait l'objet d'une soumission d'un article auprès d'une revue scientifique, à propos d'un processeur embarqué à faible consommation de puissance dont la vitesse varie à chaque cycle d'horloge.

Le processeur VSP est capable d'ajuster la période de son horloge système à chacun de ses cycles, dépendamment des instructions qui passent dans le pipeline. Le principe est de coupler le processeur Nios avec le synthétiseur d'horloge à période variable (VPCS) pour implémenter un tel processeur. Le VPCS est un circuit qui permet de changer rapidement la longueur de la période de l'horloge, à chaque cycle, sur un vaste intervalle de valeurs. La résolution obtenue dépend du nombre de phases disponibles. Ces atouts permettent d'augmenter la performance ainsi que de réduire la consommation d'énergie. Le VPCS a été synthétisé sous la technologie CMOS 180 nanomètres. Le design consomme moins de 10µW/MHz dans un intervalle de 16-250 MHz, et sa fréquence peut varier dans un intervalle de 4-250 MHz. Le processeur VSP est implémenté avec la plateforme de système embarqué de Altera®, à l'intérieur du FPGA Stratix™. Le prototype réalisé fonctionne avec une horloge système dont la période varie entre 6.8ns pour les instructions rapides et 9ns pour les instructions lentes. Avec cette méthode, l'énergie dynamique consommée par boucle de programme est réduite de 14%, mais aussi le temps d'exécution est raccourci de 3.6% comparé au processeur Nios™ original

fonctionnant à sa fréquence maximale (133MHz). Dans l'article soumis il est aussi démontré qu'il est possible de réduire l'énergie consommée en maintenant la même vitesse d'exécution du programme, un atout important pour la conception des processeurs embarqués de prochaine génération.

Réduire la consommation de l'énergie d'une certaine tâche sans allonger son temps d'exécution est un important défi. La proposition faite dans ce travail est d'optimiser la consommation de l'énergie en réduisant les temps de repos dans le circuit en utilisant une horloge qui suit de très près les moments d'opérations.

La vitesse maximale d'un processeur est habituellement déterminée par le plus long délai de son chemin critique pour une certaine tension d'alimentation, ce qui fait que l'horloge utilisée dans les circuits synchrones a une période assez longue pour rencontrer les contraintes fixées par le chemin critique. Par contre, ce chemin n'est pas nécessairement toujours utilisé à chaque cycle d'horloge, ce qui fait qu'à certains moments il y a du temps perdu en attendant la fin du cycle. Avec une estimation à chaque cycle du plus long chemin utilisé par les opérations, on est capable de minimiser les temps perdus sans toucher à la conception du circuit. Notre méthode permet d'exécuter chaque instruction en un temps de cycle au juste nécessaire au lieu d'utiliser un temps constant correspondant à l'instruction la plus lente.

Au cœur du processeur à vitesse variable se trouvent le VPCS et un circuit de contrôle pour sélectionner la valeur de la période de l'horloge à utiliser selon les instructions présentes à chaque cycle dans le pipeline. La sortie du VPCS sert d'horloge pour tous les éléments du système y compris lui-même. Seules les parties qui communiquent avec d'autres systèmes sont cadencés à une horloge fixe. Le VPCS peut multiplier la fréquence d'une horloge de référence par une valeur fractionnaire qui peut être plus grande ou plus petite que l'unité, et peut instantanément changer en un cycle d'une valeur à une autre.

4.1 Architecture du générateur d'horloge

Le VPCS génère une horloge qui sera utilisée par presque tous les éléments du système, et transforme la période d'horloge à un multiple fractionnaire de celle de référence. Un diagramme bloc représentant son architecture est illustré à la Figure 4-1, où chaque bloc est décrit dans les paragraphes qui suivent. Le diagramme temporel quant à lui est représenté à la Figure 4-2. Il est à noter que seules les transitions montantes des horloges générées sont représentées, étant donné que le coefficient d'utilisation du cycle de l'horloge n'est pas important dans notre cas, le circuit devra s'adapter si jamais il le devient. Le gros du VPCS est très similaire au DDPS, avec un générateur et un sélecteur de phases, mais le circuit tout autour est modifié dans le but de produire une large gamme de fréquences, au dessus et en dessous d'une valeur de référence, avec la capacité de changer la fréquence en un cycle.

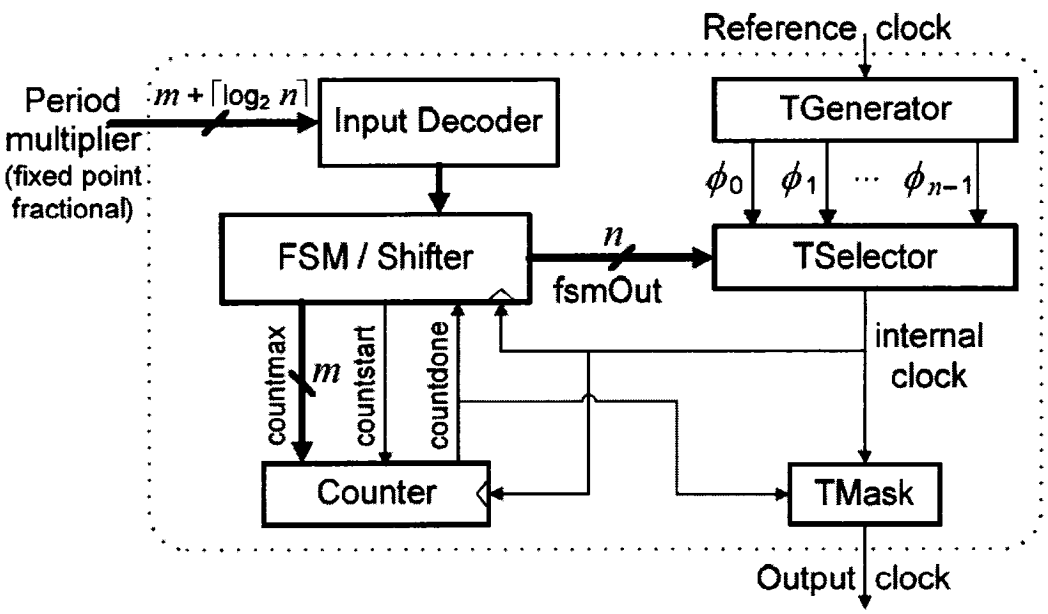

Figure 4-1 : Diagramme Bloc du VPCS.

Le multiplicateur de période se compose de deux parties : m bits pour la partie entière et $\lceil \log_2 n \rceil$ bits pour la fraction, avec n comme dénominateur. La résolution en temps des différentes périodes du signal de sortie est donnée par la période de l'horloge de référence divisée par n. Le module générateur de transition (*TGenerator*) produit n phases de l'horloge (Φ_0 à Φ_{n-1}), chacune ayant la même période que l'horloge de référence, mais déphasées et également distribuées sur toute la période : Φ_i est l'horloge de référence retardée de i/n-ième de période. Ce module peut être implémenté en utilisant un oscillateur différentiel en boucle composé d'inverseurs tampons. Notons qu'une boucle de n inverseurs, n étant impair, produit n phases, mais si il faut analyser l'ordre des phases tout au long de la boucle on a la séquence de phases suivante : Φ_0, $\Phi_{(n+1)/2}$, Φ_1, $\Phi_{(n+1)/2+1}$, Φ_2, $\Phi_{(n+1)/2+2}$, ... , Φ_{n-1}, $\Phi_{(n-1)/2}$. Le module *TGenerator* dont la faisabilité a été étudiée [11] n'est pas implémenté dans ce circuit, on assume que les phases sont disponibles.

Le module sélecteur de transitions (*TSelector*) sélectionne une des phases venant du *TGenerator*, pour choisir à quel moment la transition de l'horloge interne doit arriver. L'horloge interne est une horloge à période variable utilisée par tous les éléments du VPCS. Sa période maximale est celle de l'horloge de référence, et sa période minimale est le plus petit délai possible plus grand que le délai critique dans le module FSM/Shifter. Si le coefficient multiplicateur de la période est inférieur à l'unité, le signal de horloge interne est le même qui se propage à la sortie du VPCS, tandis que si ce coefficient est supérieur à l'unité, l'horloge interne aura une période plus petite que la période de l'horloge en sortie, ainsi les périodes formées par les transition en extra ne seront pas propagées à la sortie, étant filtrées par le module de masquage de transitions (*TMask*). Le filtrage des transitions est contrôlé par le compteur de transitions (*TCount*), qui va compter le nombre de transitions qui ne seront pas reflétées à la sortie, pour former les périodes plus longues que celle de référence. Le masquage est fait avec une simple porte *and* dont l'entrée *countdone* est active après le front descendant de l'horloge interne, pour que la prochaine transition puisse être propagée.

Le module FSM/Shifter est le cœur de la logique de contrôle du VPCS, il est implémenté en une machine à quatre états avec des sorties registrées qui contrôlent les autres éléments du VPCS. Ses entrées de contrôle proviennent du Décodeur d'entrée, qui enregistre le facteur de multiplication à la bonne phase et calcule le décalage à effectuer. Il a été décidé d'utiliser un registre de n bits chargé avec une valeur « one-hot » rotée par la valeur de la distance de phase visée grâce à un décalage barillet, au lieu d'un accumulateur de $\log_2 n$ bits modulo n suivi d'un décodeur à n sorties comme dans le DDPS [11]. Ceci simplifie la logique quand n n'est pas une puissance de 2, et les circuits de l'additionneur et du décalage ont tous le même délai qui est de $O(\log_2 n)$. Lorsque le facteur multiplicateur de période est un nombre entier, aucune rotation de bits n'est effectuée, et les transitions des périodes de la phase d'horloge choisie peuvent être comptées pour une division de fréquence, d'autre part la partie fractionnaire du coefficient spécifie le nombre de bits de décalage.

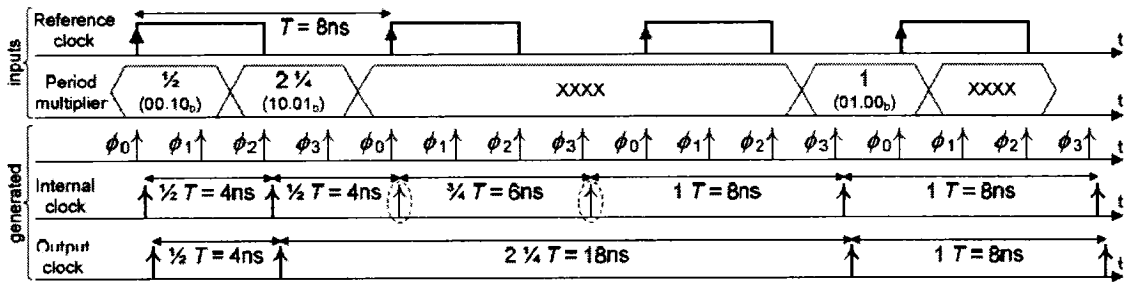

Figure 4-2 : Diagramme Temporel.

Pour comprendre pourquoi la machine à état (FSM) est un plus complexe qu'à première vue, nous expliquerons pourquoi la période 2 ¼ T de la Figure 4-2 est formée en (½ + ¾ + 1) T au lieu de (¼ + 1 + 1) T. Comme mentionnée précédemment, la période maximale interne est la période de référence T, et un compteur est utilisé si une plus longue période est nécessaire, ainsi la période 2 ¼ T ne peut pas être générée en une seule période de l'horloge interne. Aussi, la période minimale interne est contrainte par le délai du chemin critique dans la machine à état, qui est environ de 3 ns, qui représente

3/8 *T* dans notre exemple. Donc il est impossible d'obtenir une période de ¼ *T* pour former celle de 2 ¼ *T*, étant donné que ceci demande un temps de réponse de la machine à état plus court que le délai du chemin critique. Les périodes internes courtes doivent être générées seulement quand la période en sortie doit être aussi courte ; par exemple avec 16 phases on voudrait être capable de générer une période de $^{3}/_{8}$ *T*, mais celle de 1 $^{1}/_{16}$ *T* ne devrait pas générer la période $^{1}/_{16}$ *T*, sans avoir à comparer la fraction avec le délai critique réel. Pour cette raison, le module FSM est conçu de façon que les périodes de l'horloge interne ne soient jamais plus petites que la moitié de celle de référence, sauf si le coefficient multiplicateur est inférieur à l'unité. Le FSM doit donc pouvoir fonctionner à un minimum du double de la fréquence de référence. Étant donné que le FSM/Shifter est le chemin critique du VPCS et qu'il roule à la fréquence de sortie pour des fréquences supérieures à la référence, la fréquence maximale d'entrée sera la moitié de celle en sortie. Quand le multiplicateur est plus grand que 1 et sa partie fractionnaire est inférieure à ½, la période minimale désirée est garantie en avançant d'abord d'une période de ½ *T*, ensuite en complétant le reste de la période, qui correspond à ce moment à une multiplication avec fraction supérieure à ½.

La machine à états utilise les états *Integer, Fract<½, Fract>½, et Mult<1* pour contrôler les processus de décalages et de comptage selon le coefficient multiplicateur de période. Ces états déterminent la quantité de rotation pour le décalage, ainsi que la commande pour commencer le comptage et le masquage des transitions de l'horloge interne. Dans l'état *Integer*, les transitions internes sont comptées sans changement de phases, un nombre entier de périodes de référence est compté une fois que la phase désirée est choisie. Quand le multiplicateur est inférieur à l'unité, l'état *mult<1* est utilisé pour changer de phases sans compter ni filtrer. Dans ce cas toutes les transitions de l'horloge interne sont envoyées à la sortie du VPCS. Pour un multiplicateur au dessus de l'unité, avec une partie fractionnaire non nulle, on va à l'état *Fract<½* ou *Fract>½*, selon la partie fractionnaire, pour s'assurer qu'aucune période interne est inférieure à ½ *T* avant de passer à l'état *Integer* pour compter les périodes entières restantes. Un

passage à l'état *Fract*<½ se fait avec un changement de phase de ½ *T* puis un autre changement de phase correspondant au reste de la fraction est effectué avec une transition à l'état *Integer*. Tandis qu'un passage à l'état *Fract*>½ change directement de phase déterminée par la partie fractionnaire et reste à cette phase en passant à l'état *Integer* pour le comptage.

La relation entre les entrées, l'horloge interne et l'horloge de sortie est illustré dans un exemple à la Figure 4-2, avec n = 4 phases, une période de référence de T = 8ns, et une séquence de coefficients multiplicateurs de ½, 2 ¼ et 1 ; les transitions encerclées de l'horloge interne sont celles qui sont masquées donc non propagées à l'horloge de sortie.

4.2 Aperçu architectural du VSP

L'idée principale est de définir la durée d'un cycle d'horloge en particulier selon le pire cas en terme de délai de propagation d'une certaine instruction lorsqu'elle s'exécute. Chaque instruction a un chemin critique de donnée différent des autres, donc la durée du cycle varie d'un cycle à un autre. Pour un processeur pipeliné, le code de la prochaine instruction à exécuter peut être utilisé pour prédire le chemin critique du plus long étage du pipeline selon le flot de donnée de l'instruction courante. Les signaux de contrôle du pipeline du processeur sont extraits et utilisés par la logique de contrôle de l'horloge pour fixer précisément la durée du bon cycle lorsqu'on est en présence des suspensions dans le pipeline. Les délais des instructions mesurés selon les différents niveaux de voltage sont stockés dans une table et peuvent être utilisés pour déterminer pour chaque cycle la période de l'horloge requise. Ceci dans le but de réduire le temps gaspillé pour soit accélérer les opérations ou diminuer la demande en énergie. Même si on a un voltage constant avec la même vitesse moyenne que le processeur standard, le VSP peut sauver de l'énergie en utilisant un voltage réduit comme illustré à la Figure 4-3.

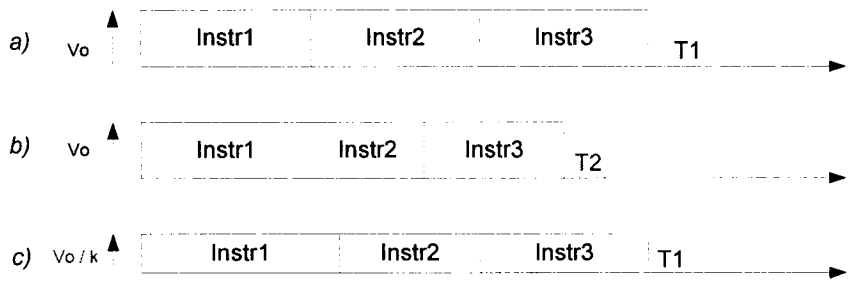

Figure 4-3 a) Application sans l'ajustement des cycles d'horloge, b) même application utilisant les cycles variables, c) l'application à la même vitesse que celle en a) , mais avec moins d'énergie et un voltage réduit.

4.2.1 Diagramme Bloc

Pour démontrer le concept, un prototype dans lequel la période de l'horloge change à chaque cycle en fonction du vrai chemin critique est développé sur une plateforme de FPGA. Ce système est basé sur le processeur Nios de Altera [2], un IP logiciel de processeur configurable dont l'accès à la description VHDL pour le générer est ouvert. Le processeur dans ce travail est configuré simplement sans mémoire cache instruction ou donnée, non optimisé pour réduire les suspensions de pipeline, et utilise la mémoire embarquée du FPGA. Les signaux nécessaires provenant du pipeline du processeur et un aperçu de la logique qui contrôle l'horloge sont illustrés à la Figure 4-4.

L'horloge générée par le VPCS est utilisée pour presque tous les éléments du système comme illustré à la Figure 4-5. Le VPCS reçoit d'autres horloges qui représentent les différentes phases disponibles de l'horloge de référence.

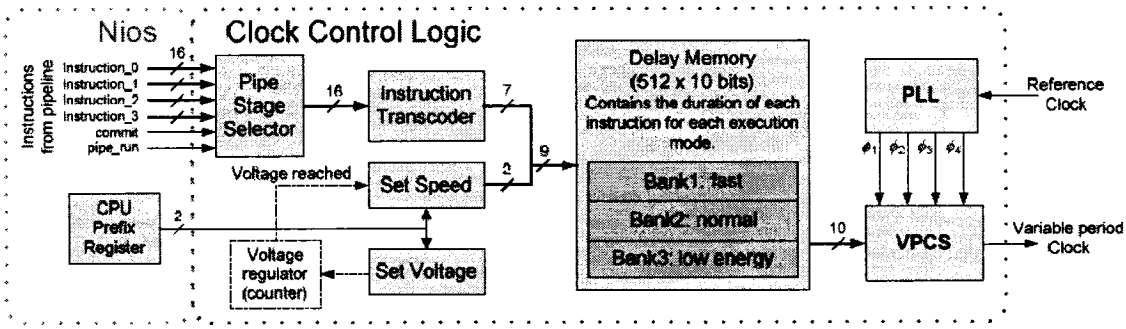

Figure 4-4 : Architecture du VSP prototype implémenté sur FPGA.

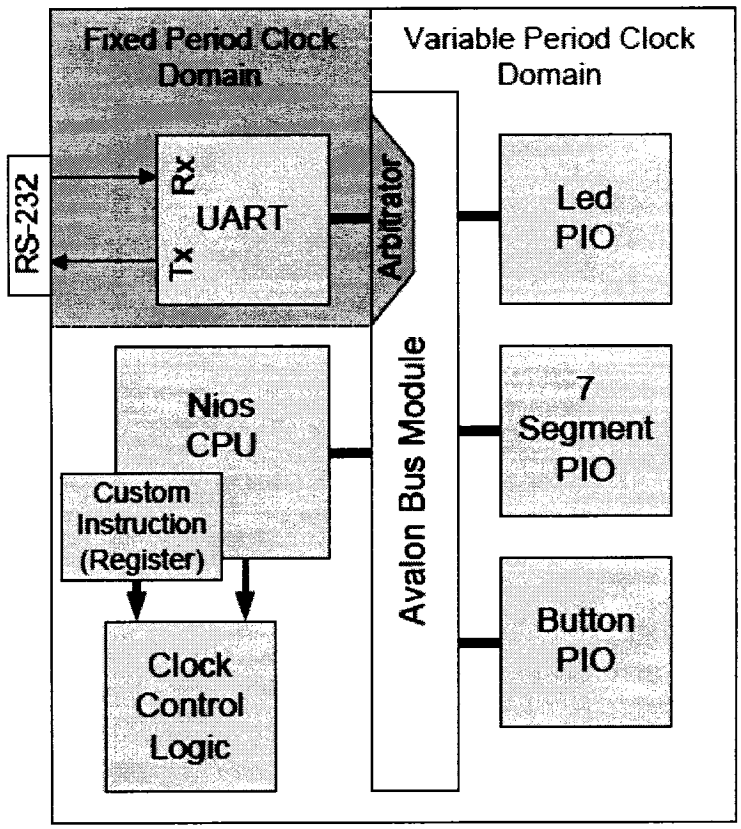

Figure 4-5 : Aperçu de la plateforme VSP.

Notre implémentation est limitée par des contraintes apportées par la plateforme de prototypage rapide comparé à si elle était faite en technologie ASIC. Une alimentation variable ou différents niveaux de tension ne sont pas disponibles sur la carte de développement, mais la logique qui supporte l'échelonnage de la tension est implémentée pour vérifier la complexité et la faisabilité, et est partie intégrante du VSP. Pour implémenter le générateur de transitions du VPCS, on utilise un PLL du Stratix™. Il peut générer quatre phases de l'horloge principale qui sont distantes également les unes des autres, ceci limite la résolution en phases de notre système. Avec une approche visant la technologie ASIC, il aurait été possible d'avoir 64 phases ou plus, le DDPS [11] est implémenté avec 16 phases dans le générateur de transitions. L'énergie et la puissance dissipées ne sont pas mesurées physiquement mais estimées à travers le logiciel après placement et routage comme indiqué au paragraphe précédent. Malgré toutes les limitations, le concept a été expérimenté physiquement sur la carte de développement.

4.2.2 La logique de contrôle

Dans le Nios, les signaux du champ des instructions utilisées sont disponibles pour chaque étage du pipeline, ils sont donc extraits et connectés au *Pipe Stage Selector* dans le but de choisir celui dont l'exécution du moment est la plus longue. Le circuit est particulier au Nios, et peut être plus complexe selon les processeurs car il s'agit ici d'évaluer la durée d'exécution de toutes les instructions dans le pipeline et de prendre la valeur maximale. Dans le Nios, seulement l'UAL et l'étage d'accès mémoire sont confrontés pour déterminer le délai de l'étage le plus long, ce qui simplifie le contrôle.
Le transcodeur d'instructions extrait le code de largeur variable du champ d'instruction et le transforme en une taille fixe (7 bits). En combinaison avec 2 bits venant du régulateur de voltage et de vitesse (*Set Speed*), il permet de faire l'adressage pour la mémoire à délais. Cette mémoire contient les coefficients multiplicateurs de période

correspondants aux différents délais des instructions, pour chaque mode d'alimentation. Le délai entre deux phases consécutives est approximativement de 2 ns selon ce qui nous est offert par les ressources en PLL du FPGA.

Une instruction spécialisée *setmode* a été implémentée dans l'UAL du CPU configurable pour choisir les différents modes de fonctionnement rapide, normal ou basse énergie, ceci en sélectionnant la banque correspondante dans la mémoire à délais. L'instruction fixe les 2 bits de poids faible du registre de préfixe dans le Nios, puis l'envoi en un cycle aux modules *Set Speed* et *Set Voltage.* Le temps de réaction à cette commande de changement de voltage peut prendre des dizaines de microsecondes avant que le niveau de tension désiré soit atteint, à ce moment le régulateur de voltage envoie un signal au module *Set Speed.* Nous utilisons un compteur à la place du régulateur de tension étant donné qu'il n'en a pas de disponible sur la plateforme utilisée. Comme illustré à la Figure 4-6.

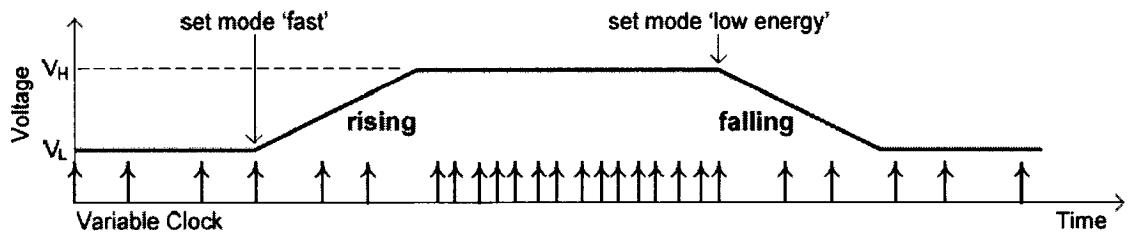

Figure 4-6 : Régulation de voltage.

Pour éviter des erreurs lors du changement de voltage, le module *Set Speed* compare le mode précédent au mode de fonctionnement désiré pour déterminer l'action à prendre. Si le mode désiré est plus rapide que celui courant, cela veut dire qu'il y a un besoin d'élever le niveau de tension, la vitesse actuelle est augmentée seulement après que la régulation de voltage est complétée. Par contre, si le mode désiré est plus lent, alors la fréquence est réduite immédiatement, ceci est sans risque puisque le système peut

fonctionner à la vitesse d'un voltage inférieur, c'est-à-dire ralentir, pendant que le voltage est entrain de diminuer.

4.2.3 Aléas du Pipeline

Pour fixer la durée du prochain cycle, le processeur doit considérer quelles instructions sont dans le pipeline. Il peut arriver des suspensions dans le flot de donnée des instructions présentes dans le pipeline, causées par les différents aléas ou les instructions à plusieurs cycles . Pour pallier à ces problèmes, nous avons extrait deux signaux additionnels du Nios. Le signal *Commit* passe au niveau bas quand l'instruction en exécution requiert plus d'un cycle d'horloge, ce signal sert à suspendre les étapes 1 et 2, de lecture d'instruction et de décodage. Le signal *pipe_run* passe au niveau bas à tout moment où on a un accès mémoire long ou un aléa de donnée, et peut suspendre toutes les étapes du pipeline.

Une fois que les instructions à exécuter sont connues, le processeur détermine quelle étape prendra le plus long temps. Dans le Nios, seulement l'UAL et les accès en mémoire peuvent avoir un délai plus long que la période minimale possible, qui est dans notre cas le délai de l'étape de décodage. Les opérations de l'UAL sont faites à la $4^{ième}$ étape et les accès mémoire à la $5^{ième}$. Lorsqu'on a une opération arithmétique ou logique, juste après un accès mémoire, les étapes numéro 4 de l'opération UAL et numéro 5 de l'accès mémoire arrivent en même temps, le sélecteur de stages choisira le stage le stage au plus long délai afin de fixer une longueur de cycle suffisante . Les tests montrent plus tard que la plupart des opérations UAL prennent plus de temps à être exécutées que les opérations d'accès mémoire.

4.2.4 Résultats de synthèse du VPCS

Le Stratix ne permettant pas d'avoir plus de quatre phases équidistantes, les tests de performance et de puissance du VPCS ont été faits en simulation avec la technologie ASIC. Trois versions du circuit différentes, surtout par le nombre de bits du multiplicateur de période en entrée, ont été testées afin de comparer l'intervalle de fréquences obtenues en sortie et la consommation de puissance. Le circuit est réalisé avec la librairie TSMC CMOS 180nm. Les paramètres *m* et *n*, listés à la table de la Figure 4-7, sont aussi définis à la Figure 4-2, spécifient la taille en bits du coefficient multiplicateur. Les deux premières versions sont des versions complètes du VPCS avec *m* =2 bits de partie entière et respectivement *n* =4 et *n* =*16* phases, la troisième version est la version réduite pour le cas où *m* =0. Les deux versions ont à *n*=4 phases ont été testées sur la plateforme du Stratix avec le VSP, mais la version réduite a été choisie à cause des contraintes du FPGA. La première ligne de chacun des trois circuits correspond à la fréquence maximale obtenue à sa sortie.

La version à 4 phases atteint une fréquence maximale de 333.33 MHz dérivée d'une entrée dont la fréquence est soit à 166.66 MHz (son double), soit à 83.33 MHz. Les deux différentes façons de générer cette fréquence ont à peu près la même consommation de puissance.

Ensuite nous comparons la première version du circuit avec la celle à 16 phases. Étant donné que le grand nombre de phases réduit la fréquence maximale à 250 MHz, les comparaisons sont faites à cette fréquence seulement. Pour produire les fréquences 250 MHz et 16.67 MHz, les deux circuits reçoivent respectivement 62.50 MHz et 15.625 MHz de fréquence en entrée. Le second circuit peut descendre jusqu'à 3.968 MHz. À la fréquence maximale, le circuit avec moins de phases consomme moins en puissance, tandis qu'à fréquence faible (16.67 MHz) c'est le circuit avec plus de phases qui consomme moins.

Circuit		Static properties — Period multiplier size		Area (mm^2)	Dynamic results — Ref. freq. (MHz)	Period multiplier	Out. freq. (MHz)	Power (mW)
		m	*n*					
Full VPCS	1)	2	4	0.0029	166.66	00.10	333.33	1.340
					83.33	00.01		1.296
					62.50	00.01	250	0.972
						11.11	16.67	0.260
	2)	2	16	0.0083	15.625	00.0001	250	2.045
						00.1111	16.67	0.163
						11.1111	3.968	0.154
Reduced VPCS	3)	0	4	0.0007	357.14	00.10	714.28	0.824
					178.57	00.01		0.810

Figure 4-7 : Caractéristiques de Synthèse ASIC du VPCS.

Normalement, en hautes fréquences le circuit du processeur consomme plus que le VPCS, donc nous sommes plus intéressés à la conception du VPCS à basses fréquences. Il consomme une puissance à peu près constante pour toutes les fréquences générées en dessous de celle de référence à cause de la machine à état qui roule à la fréquence minimum égale à celle de l'horloge de référence. Ainsi pour le mode en faible puissance et à vitesse lente, on a un meilleur résultat quand l'horloge de référence est lente et on a un plus grand nombre de bits pour la partie fractionnaire du coefficient multiplicateur. Le circuit à 16 phases consomme 37.3% moins de puissance que le circuit à 4 phases et génère à cet effet une sortie à 16.67 MHz.

Si le but est d'obtenir la fréquence la plus haute possible, une version du VPCS à peu de phases est meilleure grâce à sa faible complexité, mais si on veut obtenir la meilleure résolution de phases ou la puissance minimale de stand-by, alors un plus grand nombre de phases est requis.

Le troisième circuit est une réduction des autres versions. Il est utile dans le cas où les fréquences générées sont toutes au dessus de la référence. Avec m=0, la machine à état, le masque de transitions et le compteur peuvent être omis du design, ce qui laisse uniquement le registre de décalage comme chemin critique. Ce circuit est adéquat pour

l'implémentation du VSP dans la plateforme limitée du Stratix, d'où son utilisation dans l'expérimentation décrite plus tard. Les résultats de consommation de puissance et de performance sont inclus à la dernière partie du tableau de la Figure 4-7.

4.3 Résultats d'expérimentation du VSP et discussion

Après avoir testé tous les délais des étapes du pipeline pour toutes les instructions, nous notons que pour la plupart des instructions le délai de l'étape de décodage domine celui des autres étapes. Un plafond de 6.5 ns est utilisé pour cette raison, tandis que quelques instructions prennent 7 ns ou 7.5 ns. Les instructions les plus longues sont ADD, SUB, CMP, NEG, SWAP et les instructions de décalage.

Basé sur des données spécifiques de notre implémentation, l'accélération maximale du programme qu'on puisse atteindre dans ce cas avec notre méthode est de 7.5/6.5 (autour de 15%), avec la même consommation d'énergie dans le processeur quand aucune instruction spécialisée n'est ajoutée. Ceci semble insignifiant mais représente une preuve de concept sur un processeur qui originellement n'est pas destiné à supporter ce genre de fonctionnement où on prend avantage de l'ajustement de l'horloge sur une base cyclique.

Le banc de test expérimental consiste à exécuter une boucle d'un programme usager sur plusieurs configurations de processeur. Le programme incrémente une valeur qui se trouve à la première adresse de la mémoire de données chaque fois qu'on passe dans la boucle du programme, puis ce compteur est lu après un temps fixe afin d'évaluer la performance atteinte.

Pour valider le concept physiquement en l'implémentant sur une plateforme FPGA, combinant le VPCS avec le processeur Nios, nous avons synthétisé à nouveau le VPCS en visant la technologie du FPGA Stratix. Étant donné que les FPGA sont plus lents avec une consommation de puissance beaucoup plus élevée que les ASIC de technologie CMOS, ses performances sont diminuées. La version réduite du VPCS atteint seulement

une vitesse maximale de 200 MHz malgré l'optimisation faite au niveau du placement et routage.

Le générateur d'horloge une fois combiné avec le Nios, produit une horloge avec une période variant de 6.81 ns (146.67 MHz) pour les instructions courtes et 9.09 ns (110MHz) pour les instructions longues. Ainsi il surpasse la fréquence maximale admissible du Nios standard qui est de 133 MHz. La table à la Figure 4-8 montre le faible impact du circuit additionnel et les avantages apportés par la méthode à horloge variable. Nous proposons une métrique qui mesure l'énergie consommée par boucle pour démontrer l'avantage tiré du compromis entre la performance et l'énergie sauvée. Cette énergie (µJ/boucle) est calculée en faisant le rapport entre l'énergie consommée et le nombre de boucles exécutées dans le programme, elle est estimée par simulations. La performance quant à elle a été mesurée réellement sur la plateforme FPGA, et concorde avec les résultas de simulations. Nous obtenons un surplus en énergie par boucle de 4.25% avec le VSP sans l'application de la période variable. Par contre on note une réduction de 14% de l'énergie consommée par boucle, comparé au Nios à performance maximale lorsqu'on utilise la méthode de période variable avec deux vitesses comme mentionné dans le paragraphe précédent. Cette version du VSP roule en moyenne 3.6% plus vite. Ces résultats sont observés en utilisant le jeu d'instruction du Nios pour lequel 15% des instructions sont classées lentes, c'est-à-dire qu'elles prennent plus de 6.8ns en exécution.

4.4 Analyse d'efficacité du VSP

Les résultats expérimentaux montrent une optimisation simultanée en performance et en énergie consommée de la version à deux vitesses du VSP comparé au Nios standard roulant à sa fréquence maximale qui est de 133 MHz. Nous obtenons un gain en performance de 3.6% et on sauve 14% en énergie dynamique consommée. Avec une période d'horloge fixe, qui est la période de référence non modifiée, l'énergie

additionnelle apportée par le circuit additionnel est de 4.25% avec une surface additionnelle de 1.7%.

Nous définissons une figure de mérite (FOM) qui caractérise l'efficacité du VSP comparé au Nios standard. Il détermine l'habilité du VSP d'accélérer un programme d'exécution en dissipant moins d'énergie, même si le circuit est plus complexe. Elle est exprimée de la sorte :

$$FOM = \frac{Speedup \times EnergyRatio}{ComplexityRatio} \tag{4.1}$$

La FOM est proportionnelle à l'accélération (Speedup) et au rapport de réduction d'énergie (EnergyRatio), et inversement proportionnel à l'ajout de complexité (ComplexityRatio) qui est une pénalité de la formule. Le terme Speedup est le rapport du nombre d'instructions exécutées dans un intervalle de temps fixe, le terme EnergyRatio est le rapport d'énergie consommée par un certain nombre d'instructions, et le ComplexityRatio est le nombre des éléments logiques du FPGA utilisés. Ce sont tous des rapports entres les deux processeurs ; Nios Standard et VSP.

Notre FOM se calcule terme par terme comme suit : FOM = 1.036 x 1.14/1.017 = 1.16. Ceci étant, nous obtenons une amélioration de 16% avec notre méthode par rapport à l'utilisation d'une horloge fixe standard dans un environnement aux contraintes élevées qu'est le FPGA.

Notons que le VSP à une fréquence d'opération moyenne de 138MHz, consomme un peu moins de puissance que le Nios standard à 133MHz (1.912 W contre 2.1 W), ceci à intervalle de temps égal sans pour autant avoir eu à appliquer la variation dynamique de voltage. Un facteur non négligeable à ce résultat est le fait que le processeur VSP roulant aux fréquences 110 et 146.6MHz, permet au générateur d'horloge du Stratix de produire une fréquence relativement basse, 110MHz, consommant ainsi moins de puissance tandis qu'avec le Nios, il consomme un peu plus vu qu'il produit 133MHz.

	Target Processor				
	Nios 110MHz	Nios 133MHz	VSP 110MHz	VSP 133MHz	VSP 110MHz - 146.67MHz
Complexity (LEs)	2640	2640	2685	2685	2685
Power dissipation (W)	1.8132	2.1002	1.8849	2.1882	1.9120
Loops	435	525	435	525	544
Loop energy (µJ/Loop)	0.417	0.4	0.433	0.417	0.351
Loop energy savings	-	*reference*	-	-4.25%	13.96%

Figure 4-8 : Table de mesures expérimentales sur 100µs de simulation.

4.5 Impact des instructions spécialisées

Le Nios étant un processeur configurable, un concepteur peut créer des instructions spécialisées pour augmenter la performance de certaines applications. Très souvent ces instructions sont complexes, afin de compresser le plus possible les calculs en une instruction. Ainsi le goulot d'étranglement de l'application devient le circuit qui s'occupe de l'instruction spécialisée, non plus celui du jeu d'instruction du processeur. Cette instruction est donc adéquate pour notre méthode car elle est bien plus lente, ce qui fait qu'on en tirerait plus avantage pour atteindre une accélération plus élevée. D'ailleurs Pontikakis [31] démontre qu'avec la méthode à période variable le cycle de l'instruction spécialisée peut être étiré autant qu'on veut sans changer les durées de cycle des autres instructions. Il suffit juste d'ajouter les délais de ces instructions dans la table à délais.

4.6 Synchronisation entre le VSP et le port Série

La carte de prototypage utilise un port série RS-232 pour pouvoir supporter l'affichage pour les informations de déboguages sur un terminal. Le port est connecté à un UART

par l'intermédiaire des signaux Rx et Tx. Lors de la configuration initiale du Nios, un diviseur à rapport spécifique pour obtenir la vitesse de transfert désirée (en bits par seconde) à partir de la fréquence de référence, est introduit. Si on change la fréquence de l'horloge après cette étape initiale, on obtient une vitesse de transfert non adéquate à la communication avec le PC, le UART ne peut donc pas être cadencé par le VPCS. Pour contourner ce problème on utilise la fréquence fixe de référence pour cadencer le UART comme illustré à la Figure 4-5.

Pour pouvoir obtenir un fonctionnement correct entre les deux domaines d'horloge créés (fixe et variable), nous nous servons de la configuration du Bus Avalon du Nios qui comprend un arbitre pour chaque périphérique attaché au bus. L'Arbitre du UART utilise une communication asynchrone pour les transferts au bus, et une communication synchrone avec le UART. Donc la même horloge doit être utilisée pour le UART et son arbitre, et ce dernier peut fonctionner à une fréquence différente de celles des autres modules du bus. Cette horloge doit être de période fixe avec les mêmes caractéristiques que la configuration du système initial. Ceci permet de respecter le ratio en fréquence pour communiquer avec un terminal tout en maintenant une synchronisation correcte avec le bus.

4.7 Accélération maximale du Nios

L'utilisation des instructions lentes et rapides dans notre programme de test est assez représentative de la proportion d'instructions lentes dans le jeu d'instruction du Nios, qui est de 15%. Notre programme contient 20% des instructions qui sont de type lentes. La valeur initiale de l'accélération maximale qui puisse être tirée du Nios est environ de 15%, soit le rapport entre les vitesses lentes et rapides qui est de 7.5/6.5. Ceci n'est obtenu que dans le cas idéal où le programme utilisé n'exécute que des instructions rapides de 6.5ns de temps d'exécution.

Pour notre programme, nous ne pouvons qu'atteindre théoriquement une accélération de 12% si nous avions plus de phases disponibles sans contraintes du placement et routage. Avec seulement 4 phases, le ratio vitesses rapides sur vitesses lentes est de 4/3 avec une fréquence maximale de 147 MHz (6.8ns). Sans limitations de l'horloge du FPGA, le ratio 7.5ns/6.5ns = 15/13 peut être atteint en utilisant un VPCS version réduite avec $n \geq 15$ phases et une horloge de référence de période $0.5n$ ns. Pour générer les périodes 6.5ns, 7ns et 7.5ns avec un tel VPCS, on utiliserait un incrément de phase respectivement de 13, 14 et 15 pour atteindre l'accélération optimale.

CHAPITRE 5
Conclusion générale

Le chapitre qui suit permet de conclure cette étude en présentant de manière synthétique les trois chapitres précédents. Une brève présentation des résultats sera effectuée ainsi qu'une discussion sur les limites de nos travaux, ouvrant ainsi une brèche sur des travaux de recherche futurs.

5.1 Synthèse des travaux

Le chapitre 2 présentait une revue de littérature portant sur les travaux antérieurs effectués sur les circuits de synthèse d'horloge. Chronologiquement, l'évolution des modules de génération d'horloge est passée du simple cristal de quartz avec ses propriétés oscillantes, aux circuits modernes de synthèse d'horloge dont la plupart utilise un PLL. Le PLL étant une boucle à verrouillage de phase ayant pour rôle de stabiliser le VCO dont on peut obtenir une horloge à fréquence élevée ainsi que plusieurs phases de cette horloge. L'analyse de ces circuits montre les limites en ce qui concerne la variabilité en fréquences, étant donné que la fréquence à la sortie dépend du facteur de division. Le circuit DDS quant à lui procure un signal dont la fréquence maximale est réellement 1/3 de la fréquence de référence. D'autres méthodes de synthèse d'horloge utilisent les différentes phases extraites du VCO afin de reconstituer un signal dont la fréquence est le multiple de la fréquence de base. Cette méthode est utilisée par le DDPS et la méthode de « *flying adder* ». Ces circuits ne permettent que de multiplier la fréquence de l'horloge par un facteur fractionnaire contrairement au VPCS qui a l'avantage de pouvoir changer d'une fréquence très élevée, multiple de la fréquence de référence, à une fréquence très basse provenant de la division de l'horloge de référence.

Une revue de littérature sur les processeurs supportant le *voltage scaling* dynamique (DVS) dont la fréquence de l'horloge est ajustée dynamiquement avec le voltage, permet de situer le VSP issu de nos travaux qui est conçu pour supporter les algorithmes de DVS comme ses prédécesseurs.

Le chapitre 3 présente les méthodes et outils de conception utilisés tout au long de la réalisation de ce projet de maîtrise, ainsi qu'une description en détails de l'implémentation de notre circuit de synthèse d'horloge et du processeur à vitesse variable. La conception du VPCS en technologie ASIC requiert une méthode d'analyse différente des méthodes conventionnelles. Les outils de mesures de puissance utilisant des méthodes d'analyse statiques, le défi à relever était d'évaluer la puissance dynamique consommée par le VPCS selon les différents modes de fonctionnement. Ces derniers nécessitant une variation dynamique de la période de l'horloge de sortie. La migration du VPCS de la plateforme ASIC vers la plateforme FPGA a nécessité des modifications afin de rencontrer les spécifications voulues. Dans ce chapitre, la modification du noyau logiciel du processeur Nios® qui mène à la réalisation du VSP est décrite dans les détails ainsi que les ressources de la plateforme Stratix® utilisée. Les outils fournis permettent d'évaluer efficacement la consommation en puissance dynamique par simulation.

Dans le chapitre 4, la conception du circuit du VSP au complet est présentée et a été l'objet d'un article qui a été soumis à l'acceptation d'un journal scientifique. Le processeur à vitesse variable (VSP) est adapté aux applications à faible consommation de puissance. Le VSP change à chaque cycle, sa période d'horloge en fonction des instructions du programme en cours d'exécution, ceci en tenant compte à chaque fois du temps d'exécution requis par l'étage le plus long du pipeline. Le circuit de prédiction de l'étage le plus long prend aussi en compte les aléas et les suspensions qui arrivent dans

le pipeline. Les instructions longues sont exécutées sans pour autant rallonger les temps d'exécution des instructions courtes, ce qui procure une accélération du VSP par rapport aux processeurs standard utilisant une période d'horloge fixe, dont la vitesse correspond au délai d'exécution des instructions les plus longues. Malgré cette accélération, le surplus d'énergie consommée est faible. Une implémentation efficace permettrait de réduire considérablement l'énergie consommée en réduisant le voltage tout en conservant les objectifs de performance similaires au processeur non accéléré. Ce concept peut être combiné à la méthode de DVS pour réduire aussi l'énergie consommée relativement à la demande en performance des applications.

Une étude est faite dans cet article pour mettre en exergue la contribution de l'accélération apportée par le design du VSP combinée avec le gain en consommation d'énergie. La métrique utilisée pour le calcul énergétique est l'énergie par boucle de programme (Joules/loops) qui est l'équivalent inverse d'une autre métrique utilisée pour certains processeurs [41] embarqués qui est le MIPJ (MIPS par Joule).

Nous avons proposé une figure de mérite qui valorise le facteur d'accélération et le gain en énergie du VSP face à un processeur Nios standard à vitesse maximale, en tenant compte de la pénalité apportée par l'ajout de la circuiterie pour réaliser la période variable (énergie, complexité, effets de placement et routage). Les mesures expérimentales effectuées permettent d'obtenir une figure de mérite de 16%.

Un prototype du VSP a été implémenté sur la plateforme de développement de système embarqué d'Altera®, à l'aide du processeur embarqué Nios et des ressources du FPGA Stratix. Le VPCS intégré au Nios est réduit pour produire une période de durée 9.09ns pour instructions longues et 6.8ns pour instructions courtes, ainsi le processeur fonctionne à 110 MHz et 147 MHz. Le système composé de l'ensemble du FPGA consomme en tout 1.912 Watts configuré avec le VSP, une consommation qui est 276 mW moindre que le système simplement configuré avec le Nios fonctionnant à sa vitesse maximale (133 MHz).

5.2 Indications de recherches futures

Les résultats obtenus dans le chapitre 4 sont satisfaisants car ils démontrent la possibilité de économiser de l'énergie en utilisant la méthode proposée tout en accélérant l'application.
Cependant, cette accélération n'est pas très élevée à cause de la faiblesse des écarts entre les délais de traitement des instructions. Néanmoins l'évaluation des délais des instructions n'est pas efficace car le délai de l'étage de décodage du pipeline est dominant pour la plupart des instructions, ce qui nous force à attribuer la valeur du délai de l'étage de décodage à ces instructions. Des travaux futurs pourront être effectués sur les processeurs dont l'architecture pourra tirer avantage d'une horloge à période variable.

La méthode explorée dans ce mémoire est plus efficace pour les processeurs configurables avec des instructions spécialisées utilisant des ressources matérielles dédiées pour leur exécution, que pour les processeurs standard. En effet il suffit d'imaginer un processeur avec ne serait ce qu'une seule instruction spécialisée très longue, qui pénaliserait le reste des instructions. Le profilage d'un tel processeur lui attribuerait une fréquence maximale correspondant au délai de cette instruction, tandis que la méthode d'horloge à période variable permettrait d'ajuster la période seulement au cycle d'exécution de l'instruction en question. Ainsi on obtiendrait une accélération plus consistante. Une expérience plus approfondie devrait être faite avec plusieurs exemples de programmes car cette accélération dépendrait de l'occurrence des instructions spécialisées ainsi que de leur temps d'exécution moyen. Les travaux de Pontikakis [31] seraient par la même occasion validés expérimentalement.

D'autre part, le VPCS n'a pu être configuré que pour 4 phases puisque les PLLs du Stratix™ limitaient à 4 le nombre de phases équidistantes disponibles. Ceci nous offrait

une densité de phase faible, donc une résolution en fréquence également faible. Avec plus de phases on a la possibilité d'obtenir des fréquences élevées pour une fréquence de référence basse, d'où une augmentation de la plage de fréquence (actuellement à FMax / Fmin = 1000), ce qui est un avantage pour les systèmes de faible puissance.

La limitation majeure qui empêche non pas de prouver mais de valider le concept vient du fait que la plateforme de développement n'offre pas la possibilité de réduire le voltage, ce qui apporterait une meilleure réduction de l'énergie. Un développement plus approfondi conduirait à un processeur embarqué à très faible consommation de puissance.

Bibliographie

[1] Abramovitch,D. 2002. "Phase-Lock Loops: A Control Centric Tutorial". Proceedings of the 2002 ACC. Article consulté

[2] Altera Corporation. http://www.altera.com.

[3] Blanchard, A.1976. "Phase-Locked Loops". New York, NY: John Wiley & Sons.

[4] Benini, L., Michelli, G. Aug. 1999. "System-level power optimization: Techniques and tools". Proceedings International Symposium of Low Power Electronics & Design. Pages 288–293.

[5] Best R. E. 1997. "Phase-Locked Loops: Design, Simulation, and Applications". New York: McGraw-Hill, third edition.

[6] Bozic, M., Gardiner, J.G.1993 "Direct Digital synthesis spurious noise properties" IEE Colloquium on Characterization of Oscillators – Design and Measurement (Digest No. 025). Pages 1/1-6.

[7] Brennan P.V. 1996. "Phase-Locked Loops: Principles and Practice". New York: McGraw Hill.

[8] Brynjolfson I., Zeljko Z. 12-15 Sept. 2001. "MCSoC: A platform for Clock Managed Systems on a Chip" ASIC/SOC Conference, Proceedings of 14th Annual IEEE International. Pages: 117-121,

[9] Brynjolfson, Zilic Z. May 2000. "Dynamic Clock Management for Low Power Applications in FPGAs," Proceedings of IEEE Custom Integrated Circuits Conference. Pages:139-142.

[10] Burd, T.D., Pering T.A, Stratakos A.J., Brodersen R.W. Nov. 2000. "A dynamic voltage scaled microprocessor system", IEEE Journal of Solid-State Circuits. Pages:1571-1580, Volume: 35, Issue: 11.

[11] Calbaza, D.E.; Savaria, Y. Aug. 2002. "A direct digital period synthesis circuit"; IEEE Journal of Solid-State Circuits, Volume 37, Issue 8, Pages:1039-1045

[12] Calbaza, D.E.; Savaria, Y. March 2001. "Direct digital frequency synthesis of low-jitter clocks" IEEE Journal of Solid-State Circuits, Volume 36, Issue 3, Pages: 570-572

[13] Chun-Huat Heng; Bang-Sup Song. June 2003. "A 1.8-GHz CMOS fractional-N frequency synthesizer with randomized multiphase VCO" , IEEE Journal of Solid-State Circuits, Volume 38, Issue 6, Pages: 848-854

[14] Crawford J. A. 1994. "Frequency Synthesizer Design Handbook". Norwood, MA 02062: Artech House.

[15] Cooper H.W. 19th July 1974. "Why complicate frequency synthesis", Electronics Design 15. Pages: 80-84

[16] Filiol N, Riley T.A.D., Plett C., Copeland M.A. July 1998. "An agile ISM band frequency synthesizer with built-in GMSK data modulation," IEEE Journal of Solid-State Circuits, vol. 33, Pages: 998–1008,

[17] Gardner F. M, "Phase lock TechniquesÉ. New York, NY: John Wiley & Sons, second ed., 1979. ISBN 0-471-04294-3.

[18] Garvey J.F.; Babitch D. 1990. "An exact spectral analysis of a NCO based Synthesizer" Proceedings of the 44th Annual Frequency Control Symposium. Pages: 511-521

[19] Goldberg B. – "Digital techniques in frequency synthesis" New York : McGraw-Hill, 1996

[20] Grunwald D., Levis P., Farkas K.I. Oct. 2000. "Policies for dynamic clock scheduling". Proceedings of Symp. Operating Systems Design and Implementation (OSDI), San Diego, CA.

[21] Gupta, S.C "Phase-locked loops," Proceedings of the IEEE, vol. 63, pp. 291-306, February 1975.

[22] IBM PowerPC, http://www.chips.ibm.com/power/powerpc/

[23] Intel Xscale, http://www.intel.com/dsign/intelxscape/

[24] Kuroda, T. & AL. Mar. 1998. "Variable supply-voltage scheme for low-power high-speed CMOS digital design" IEEE Journal of Solid-State Circuits, vol. 33, Pages: 454–462.

[25] Lindsey W.C., Chie C.M. April 1981. "A survey of digital phase-locked loops" Proceedings of the IEEE, vol. 69, pp. 410–431,.

[26] Lindesy W.C., Simon M.K. 1978. 'Phase-Locked Loops and Their Application". IEEE PRESS Selected ReprintSeries, New York, NY: IEEE Press.

[27] Lindesy W.C., Chie C.M.. 1986. Phase-Locked Loops. IEEE PRESS Selected Reprint Series, New-York, NY: IEEE Press.

[28] MACII, E. & AL. November 1998. "High-Level Power Modeling, Estimation, and Optimization" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 11.

[29] Mair, H.; Liming Xiu, "An architecture of high-performance frequency and phase synthesis" Solid-State Circuits, IEEE Journal of Volume 35, Issue 6, June 2000 Page(s):835 – 846

[30] Nowka, K.J. & AL 'A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling' Solid-State Circuits, IEEE Journal of, Pages:1441 – 1447, Volume: 37 , Issue: 11 , Nov. 2002.

[31] Pontikakis B, Boyer F-R., Savaria Y. 2005. "Performance Improvement of Configurable Processor Architectures Using a Variable Clock Period", IEEE Int. Workshop on System On Chip, July 20-24.

[32] Pouwelse J. & AL. Oct 2003. "Application-directed voltage scaling", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Pages: 812 – 826, Volume: 11, Issue: 5.

[33] Razavi B. 1996. "Monolithic Phase-Locked Loops and Clock Recovery Circuits: Theory and Design". IEEE PRESS Selected Reprint Series, New York, NY:IEEE Press.

[34] Rhee W, Song B, Ali. A. Oct. 2000. "A 1.1-GHz CMOS fractional-N frequency synthesizer with a 3-b 3rd-order modulator," IEEE J. Solid-State Circuits, vol. 35, pp. 271–350.

[35] Riley T.A.D, Copeland M.A, Kwasniewski T.A. May 1993. "Delta–Sigma modulation in fractional-N frequency synthesis," IEEE J. Solid-State Circuits, vol. 28, Pages: 553–559.

[36] Simunic, T., Benini L., Acquaviva A., Glynn P., De Michelli G. 2001. "Dynamic voltage scaling for portable systems," in Proc. Design Automation Conf.

[37] Stephens D. R. 2002. "Phase-Locked Loops for Wireless Communications: Digital and Analog Implementation". Understanding Science and Technology, Boston/Dordrecht/London: Kluwer Academic Press, second ed.

[38] Tierney J,Rader C., Gold B. March 1971. "a Digital Frequency Synthetizer" IEEE transactions on Audio and Electroacoustics , vol. Au-19, no.1, Pages 48-56

[39] Transmeta Crusoe, http://www.transmeta.com/

[40] Viterbi A.J. 1966. "Principles of Coherent Communication". McGraw-Hill Series in Systems Science, New York, NY: McGraw-Hill.

[41] Weiser M, Welch B, Demers A, Shenker S. November 1994. "Scheduling for reduced CPU energy". In Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI), pages 13–23.

[42] Wolaver D. H. 1991. "Phase-Locked Loop Circuit Design". Advanced Reference Series & Biophysics and Bioengineering Series, Englewood Cliffs, New Jersey 07632: Prentice Hall.

Annexes

ANNEXE A

A Variable Period Clock Synthesis (VPCS) Architecture for Next-Generation Power-Aware SoC Applications

François-R. Boyer*, Habib G. Epassa*, Bill Pontikakis†, Yvon Savaria†, Wei Ling†

*DGI / †DGE, École Polytechnique de Montréal
C.P. 6079, Succ. Centre-Ville, Montreal, Quebec, Canada, H3C 3A7
E-mail : { *Francois-R.Boyer, HabibDGabriel.Epassa, Bill.Pontikakis, Yvon.Savaria, Wei.Ling* }*@PolyMtl.ca*

ABSTRACT

This paper, presents a Variable Period Clock Synthesis (VPCS) architecture that has the ability to multiply or divide a reference clock frequency on the fly, depending on the application requirements. The VPCS architecture has the advantage of switching from a current clock frequency to a target one within only one clock cycle thus, improving frequency switching delays compared to previous designs. The VPCS design also has the ability to generate any period with any resolution, an important feature that saves power in devices with multiple frequency requirements. A prototype of the VPCS architecture was developed in VHDL and synthesized in CMOS 0.18 µm technology. The design generated clocks with frequencies up to 333.33 MHz. A design aiming at a maximum frequency of 250 MHz will have a low power clock generation of 0.16 mW when running at 16.67 MHz, using 16 phases of a 15.625 MHz reference clock. This design is suitable for high-speed, energy-efficient portable applications with variable speed needs.

1. INTRODUCTION

The latest market growth of mobile battery-powered electronic devices such as PDAs, digital cameras, and mobile phones, has set new goals in the design of systems-on-a-chip (SoCs) for the next generation applications. These goals include the need for high-speed, energy-efficient, and power scalable SoC designs that would provide high-performance electronic devices with prolonged battery life. Since power consumption is directly proportional to frequency, by dynamically adjusting the clock period we are able to save on dynamic power dissipation and therefore, prolong battery life.

Recent development of low power SoC design styles and digital video and communication applications require dynamically adjustable low power clock solutions. To address these requirements, we have designed the Variable Period Clock Synthesis (VPCS) architecture. The VPCS architecture can act as a frequency multiplier or a frequency divider on the fly, depending on application requirements. The circuit can multiply or divide a reference clock frequency with an integer, fractional, or a combination of integer-fractional number, as shown in Fig. 1. Thus, VPCS can generate any desired period with any desired resolution within one clock cycle, giving great advantages in terms of power reduction in devices with multiple frequency requirements, while minimizing frequency switching delays. The power savings of VPCS are prominent in devices that do not need a high frequency running at all times.

In our work, a prototype of the VPCS architecture was developed in VHDL and synthesized in CMOS 0.18 µm technology, using Synopsys and Mentor Graphics tools. The prototype VPCS architecture generated clocks with frequencies up to 333.33 MHz. Its estimated cell area count is 0.0029 mm^2, while its estimated power dissipation is 1.340 mW when running at the maximum frequency using 4 phases of the reference clock. That prototype can produce clocks with a finest clock period resolution of 1.5 ns when using four different phases of its maximum reference clock frequency at 166.66 MHz. These results are obtained from simulation after synthesis. At high speeds, a processor driven by this clock would consume a lot more power than the clock generator and therefore, the VPCS results that are more important for the power reduction are those achieved at low speed operation. A design aiming at a maximum frequency of 250 MHz, but using low power at lower frequencies, may consume only 0.16 mW of dynamic power when generating clock frequencies from 4 to 16 MHz. These frequencies are created from 16 phases

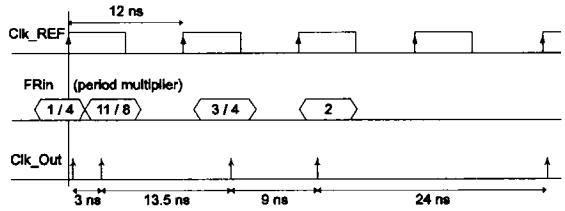

Fig. 1. A timing diagram demonstrating VPCS

of a 62.5 MHz reference clock. In these results, the power of the clock phases generator (named TGen on Fig. 2) is not included. The resolution of possible clock periods increases when the number of clock phases increase, and in our preliminary study, we achieved a finest resolution of only 375 ps, while using 16 phases of a reference clock period at 166.66 MHz.

This paper is organized as follows. In section 2 we present previous work related to the aspects addressed in this paper. In section 3, we present the architecture of the VPCS and discuss its operation. In section 4, we present the results of our synthesized architecture. Finally, in section 5, conclusions and recommendations for future research are presented.

2. RELATED WORK

Clock multiplication and clock division are commonly used to help distributing high speed clock. One approach to low power SoC design is the Globally Asynchronous and Locally Synchronous (GALS) approach. This design style demands low power clock generator or clock multiplier to be integrated into locally synchronous modules. In addition, due to asynchronous communication between different modules, the local clock should be stretchable to prevent metastable problems. The stretchable clock is realized using a pausable clock together with a delay line to reactivate the clock. Detailed studies can be found in [1]-[3]. The disadvantages of this approach are that the clock is not adjusted digitally, it is less controllable, and it takes time to be reactivated. Our clock is easily stretchable on a per cycle basis.

Another power reduction technique, Dynamic Voltage Scaling (DVS), trades-off processor frequency and hence, performance, against power consumption. A simple counter based frequency divider was used in [4] to deliver an output frequency lower than a reference frequency. The clock generators used in DVS have high accuracy, but their operating frequency range is limited, and they are not energy efficient. For the clock generation, since VPCS can generate frequencies above the reference value, we show that using a lower frequency reference reduces the power consumed when running at low frequency, and does not increase considerably the power when running at high frequency.

In multi-standard digital television and telecommunication applications, transferring from one standard to another may require a frequency multiplication or division by a fractional number. A fully integrated Direct Digital Period Synthesis (DDPS) circuit was proposed in [5] for these kind of applications. The DDPS architecture can multiply a reference clock frequency with any fractional number.

The clock generator proposed in this paper, VPCS, can adjust the frequency using any resolution within the same clock period unlike DDPS. In addition, VPCS does not need to turn off the clock like GALS designs need and therefore, does not have wake up delays. Finally, VPCS supports a wider frequency range than previous designs, based on frequency multiplication and division.

3. SYSTEM ARCHITECTURE

The block diagram of the VPCS architecture is depicted in Fig. 2. As an example, a design with only four clocks is used, but the number of clocks can be increased, depending on the period resolution that is needed. In the diagram, the transition generator block (TGen), produces several clocks each having the same frequency as the reference input clock (Clk_REF). The clocks are delayed from each other by a constant phase equal to ¼ of the reference period. The transition selector block (TSel), selects the different phases coming from the TGen block, and propagates them to the internal control clock (Ctrl_Ck). The transition counter block (TCount), when needed, counts the transitions for frequency division. The transition mask block (TMask), masks unneeded transitions coming from Ctrl_Ck, in order to obtain the desired output clock (Clk_Out). A four-state Finite State Machine merged with a shifter (FSM/Shifter) controls the TSel block, the TCount activation or deactivation, and the TMask block. As with DDPS [5], VPCS has the ability to select the transition that propagates to the output clock, and therefore, can numerically control its output period.

The transition generator circuit, TGen, can be implemented using a differential ring oscillator composed of buffers/inverters. In this work, the transition generator circuit was not implemented; instead, it is assumed that the clocks are available.

The internal control clock, Ctrl_Ck, is used to control the FSM and the transition counter, TCount, in order for a new phase selection or a new period count to start. From this clock the output clock, Clk_Out, is derived.

The TMask block masks the unneeded pulses from Ctrl_Ck in order to obtain the output clock signal,

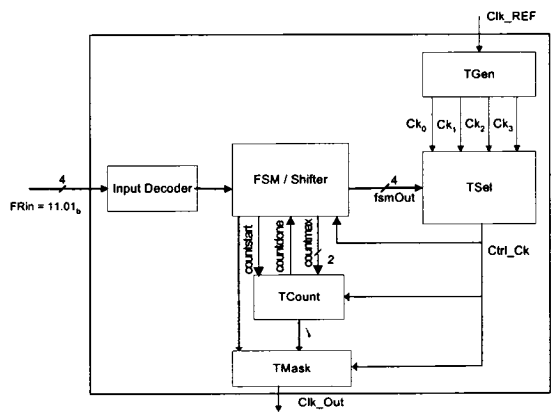

Fig. 2. Block Diagram of the VPCS

Clock_Out. This feature is illustrated in Fig. 3, where the circled arrow in Ctrl_Ck represents an internal clock transition that is masked and therefore, not shown in the output clock Clk_Out.

The role of Input Decoder is to provide the FSM the desired input phase and to compute the next shifting value of the Shifter.

The FSM/Shifter block is the core control logic of the design and is implemented as a four-state finite state machine with clocked outputs. These outputs are the Shifter registers, the CountMax registers, and the CountStart register, as shown in Fig. 2. The FSM is used to control the TSel, TCount, and TMask blocks. In other architectures, such as in [5], a phase-accumulator is used to control the phase increments and external clocks.

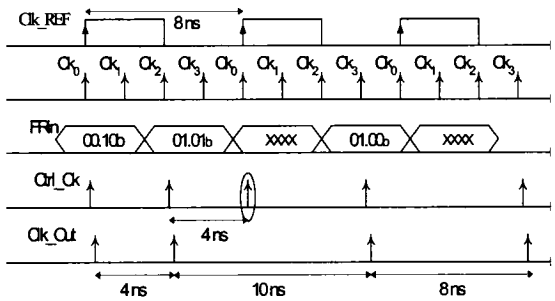

Fig. 3. Period multiplier change, from only fractional to integer and a fractional number, to only integer number

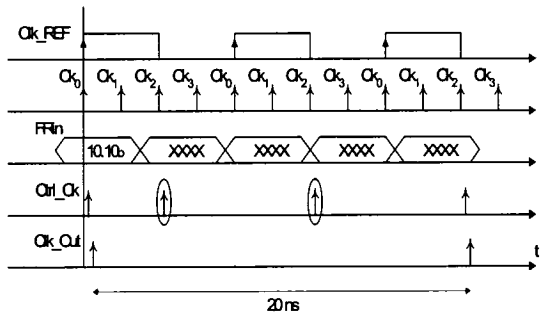

Fig. 4. Period multiplied by a fraction bigger than unity

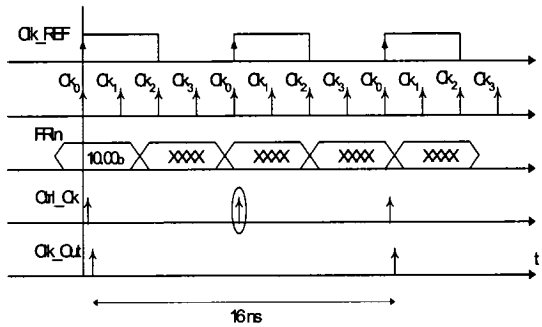

Fig. 5. Period multiplied by an integer: clock division

4. SIMULATION RESULTS

When the phase delay between two clock transitions Ck_i and Ck_{i+1} is too small, in our example this would require to react in 2ns, the FSM is not able to select the next clock transition (i.e. Ck_{i+1} cannot be selected if Ck_i is the current clock transition), due to delay limitations inside the FSM/Shifter circuit and thus, the transition is completed in two steps. Initially, it is guaranteed that the FSM can select the clock transition with a phase delay of at least one-half from the clock reference, Clk_REF, (i.e. $Ck_{i+N/2}$ can be selected if Ck_i is the current clock transition and N is the number of clock phases) and afterwards, the target transition is selected to complete the frequency switch (i.e. Ck_{i+1} is selected after $Ck_{i+N/2}$ was initially selected). This guaranteed minimum clock transition selection is called "half transition".

This feature is illustrated in the timing diagram depicted in Fig. 3. In the figure mentioned, FRin represents the input period multiplier (or frequency divider) factor in binary representation. In this case the period multiplier factor is 00.10_b (0.5_{10}) initially, which represents a clock frequency multiplication by two, and therefore, the output clock, Clk_Out, has a clock twice as fast as the clock reference, Clk_REF. The period of Clk_Out is 4 ns since the period of Clk_REF is 8 ns in this case. Assuming the next requested period is 01.01_b (1.25_{10}) later on, instead of jumping 1.0_{10} first and then 0.25_{10} to select the output period factor of 1.25_{10}; we first select 0.5_{10}, the "half transition", and then another 0.75_{10} for a total transition of 1.25_{10}. The "half transition" phase selection is shown with the circled arrow of Ctrl_Ck signal and the final phase of 1.25_{10} is the next arrow to the right. As it can be seen in Fig. 3, the circled arrow is masked in the Clk_Out signal and the final period is 10 ns (1.25 x 8 ns). Finally, Fig. 3 shows a period selection with only an integer number, 01.00_b, which is just a multiplication by unity, and therefore, the period of Clk_Out is the same as the period of Clk_REF (8 ns).

In Fig. 4, the period multiplier factor request is 10.10_b (2.5_{10}), which sets the next target pulse 2.5 times longer than the Clk_Ref period. Thus, the Clk_Out signal has a period of 20 ns (2.5 x 8 ns). This is the case were the FSM can safely choose directly the next phase, since the "half transition" requirement for clock transition is met.

Finally, Fig. 5 illustrates the case where the period selection factor is 10.00_b (2.0_{10}). Since this case has only an integer part, it represents a clock division of the reference clock Clk_REF. Consequently, Clk_Out has a period of 16 ns (2 x 8 ns).

We tested two different circuits with different bit count for the period selection in order to compare the power consumption and frequency range. The VPCS prototypes were developed in VHDL and results given by simulation after synthesis in CMOS 0.18 μm are shown in Table 1. The first four columns state static properties of the circuit, where Int and Fract represent the number of bits in the integer and fractional part of the period selector factor FRin, respectively, Ph., represents the number of phases, and Area is the area occupied by the design. The last four columns represent different modes of operation for the circuit, where f_{Clk_ref} is the reference clock frequency, FRin, is the period multiplier (in binary), f_{Clk_out}, is the resulting output frequency, and Power is the power consumed at that frequency.

As shown in Table 1, the circuit with 4 phases was able to achieve a maximum output frequency of 333.33 MHz with a maximum input frequency of 166.66 MHz. The different ways of generating the maximum output frequency consume about the same amount of power, but the results that are interesting are found when running the circuits at low frequencies. Since the circuit with 16 phases operates with a maximum output frequency of 250 MHz, we tested both circuits setting that speed as the upper bound of the clock generator for comparison.

When the two circuits are operating with an input clock (f_{Clk_ref}) of 62.50 MHz and 15.625 MHz respectively, they can both produce the exact same output frequencies between 16.67 MHz and 250 MHz, but the second circuit can also produce frequencies down to 3.968 MHz. When the circuits are running at high speed (250 MHz), we see that the circuit with the least number of phases consumes less power, and when running at low speed (16.67 MHz) it is the circuit with more phases that consumes less power. Since at high speed the processor controlled by that clock will consume a lot more power than the clock generator, then for a low power mode is better to have a slow input clock and a higher number of fractional bits for the multiplier, but at the cost of a larger area. The circuit with 16 phases uses 37.3% less power than the circuit with 4 phases, to generate the 16.67 MHz output.

In the second circuit the power is almost constant when running at speeds from 4 MHz to 16 MHz, as could be expected, since the controlling FSM never runs slower than the reference. At higher frequencies, the FSM is running at the speed of the output clock frequency and thus consumes more power.

If the design goal is to achieve the maximum possible frequency, a circuit with fewer phases is better due to minimum complexity, but if the design goal is to achieve the best resolution or the minimum standby power then, the increase of phases is desired.

Table 1. Performance Characteristics of VPCS

Static properties				Dynamic results			
FRin (bits)		Ph	Area (mm^2)	f_{Clk_ref} (MHz)	FRin	f_{Clk_out} (MHz)	Power (mW)
Int	Fract						
2	2	4	0.0029	166.66	00.10	333.3	1.340
				83.33	00.01	3	1.296
				62.50	00.01	250	0.972
					11.11	16.67	0.260
2	4	16	0.0083	15.625	00.0001	250	2.045
					00.1111	16.67	0.163
					11.1111	3.968	0.154

5. CONCLUSION

In this work, we introduced the Variable Period Clock Synthesis (VPCS) architecture, which has the ability to multiply or divide the reference clock frequency with a fractional number. The VPCS architecture can dissipate very low power when operating at low frequency and can switch from a current clock frequency to a target one within only one clock cycle thus, improving frequency switching delays compared to previous designs. The VPCS design also has the ability to generate any period with any resolution and each period can have a different length if desired, an important feature that saves power in devices with multiple frequency requirements. The synthesis results show that the circuit can achieve a maximum output clock frequency of 333.33 MHz while dissipating only 1.340 mW of power. When the number of phases is increased, the design dissipates only 0.154 mW of dynamic power at 4 MHz output speed, and have an output frequency range going up to 250 MHz. The VPCS architecture can be used in next-generation battery-powered SoC devices, where power consumption and thus, power scaling, is a critical factor in the performance of the digital device. This design is mainly a proof of a concept and is not fully optimized. In future work we are planning to optimize the design and reduce the complexity of high phase count circuits for even lower power consumption at low speed.

REFERENCES

[1] P.Nilsson et al., "A monolithic digital clock-generator for on-chip clocking of custom DSP's," *IEEE J. Solid-State Circuits*, vol.31, pp. 700-706, May 1996.

[2] T. Olsson et al., "A digitally controlled low-power clock multiplier for globally asynchronous locally synchronous designs," *in Proceedings of IEEE International Symposium on Circuits and Systems*, Geneva, vol. 3, pp. 13-16, vol.3,

May 28-31, 2000.
[3] T.Olsson and P.Nilsson, "Portable digital clock generator for digital signal processing applications," *IEE Electronic Letters*, vol. 39, no. 19, pp. 1372-1374, September 18, 2003.
[4] S.Dhar et al., "Closed-loop adaptive voltage scaling controller for standard-cell ASICs," *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'02)*, Monterey, California, United States, pp. 103-107, August 12-14, 2002.
[5] D.E.Calbaza and Y.Savaria, "A direct digital period synthesis circuit," *IEEE J. Solid-State Circuits*, vol.37, no. 8, pp. 1039-1045, August 2002.

ANNEXE B

Implementation of a Cycle by Cycle Variable Speed Processor

H.G. Epassa, F.R. Boyer, and Y. Savaria
Microelectronics Research Group
Ecole Polytechnique de Montreal, Departments of electrical and computer engineering
Montreal, Canada
{ HabibDGabriel.Epassa, Francois-R.Boyer, Yvon.Savaria }@PolyMtl.ca

Abstract— **This paper presents an automatic variable speed processor (VSP) with the ability to change its clock rate on a cycle by cycle basis, according to program instructions being in the pipeline. To demonstrate the concept, we are using an Altera Nios Processor coupled to a Variable Period Clock Synthesizer (VPCS) that is used as our variable speed clock generator. The clock period variations give a speedup, with little impact on energy consumption, and that speedup can be traded for energy reduction using voltage scaling. Our proposals are supported with a prototype implemented on the Altera Embedded System Development Board that embeds a Stratix FPGA.**

I. INTRODUCTION

Research and development of real-time microprocessors, with low energy consumption, is important since there is a growing need to prolong battery autonomy as well as reaching high performance in portable devices.

Most of these researches require the support from operating systems (OS) to optimize system power, using variable speed processors (VSP) that can reduce their energy consumption by reducing the clock frequency along with the supply voltage, when real-time deadlines do not require top speed. This technique is called Dynamic Voltage Scaling (DVS) or Dynamic Frequency Clocking Scheduling (DFCS) [1].

The challenge in real-time low energy embedded processor design is to reduce the energy consumed by a certain task without increasing its execution time. This is equivalent to being able to execute the same task, with the same energy, but faster, since reducing energy and speed at the same time is "easy" (to a certain extent).

Every microprocessor can run at a maximum speed which is determined by its longest critical path delay, and which depends on the supply voltage. In current synchronous designs, for each voltage, the clock period is of fixed duration, long enough for the critical path. Since the critical path of the whole circuit is not necessarily activated at each clock cycle, time is often wasted waiting for this unused path to finish.

Our method proposes to increase the performance, without increasing energy consumption, by reducing the wasted time with a cycle by cycle estimation of the longest used path. Each instruction will run with a different cycle time, without data loss or corruption. Then we no more run at the slowest constant speed for all instructions. We identify, by running timing simulation at the typical case condition, the specific cycle period needed to run each instruction.

Taking a program running a fixed number of instructions, our method enhances the performance of such a processor while keeping the energy required to complete the task roughly the same, since the number of transistor switches is not increasing, except for our added controlling circuit.

A good designer can take advantage of that by

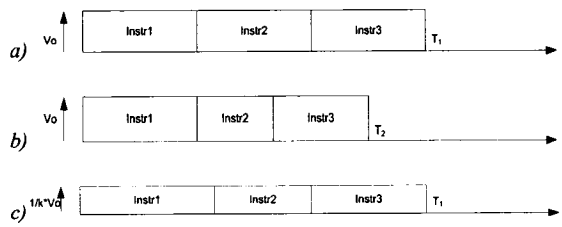

Figure 1. *a)* without cycle by cycle clock period adjustment, *b)* Faster, same energy, *c)* Same speed, less energy

scaling down the supply voltage and mean frequency at a point where the execution time for the program in the new machine remains the same as in the previous one, having the appearance of running at the same number of instructions per unit of time. This is illustrated in Fig. 1.

Our proposed architecture is intended to support DVS scheduling algorithms and also operating conditions variation, which are not demonstrated in this work. This can be done with a suitable scheduler to control the VSP circuit with a special instruction to set the desired operating mode.

This paper is organized as follows. An overview of related work is done in Section II. In Section III, we describe our approach, and present experimental results that validate our proposals in Section IV. Then we conclude in Section V.

II. RELATED WORK

Cycle by cycle clock period change has already been shown useful for enhanced performance and reduced energy consumption [1]. They use an algorithm which gives a static schedule for a data flow graph, during high-level synthesis. It produces a custom circuit with different voltages for different operational units to follow the given schedule with optimized energy consumption.

Clock self-tuning for "maximum" frequency is presented in [2]. They do it by periodically pausing computation for up to 68 cycles, during which special bit patterns are sent as ALU inputs to have the longest possible computation (the longest critical path). The output of the adder is checked after a certain delay, and depending if the result is correct or not the cycle length is decreased or increased. In [3], a Uniprocessor can enhance its performance by adapting its clock with typical conditions (temperature, manufacturing process...) by detecting timing errors using two wired copies of the pipeline. Each of those previous methods uses the same clock period, which is their critical path delay, for all operations. That critical path may not be used every time since each operation data may not pass trough that path. For example, in many processor designs, a MOV operation do not use the same ALU logic as an ADD, the latter using a longest path to refresh the carry chain bit. Our architecture is capable of selecting the maximum delay of the blocks actually used in the processor pipeline for each operation.

Several other methods have been proposed in order to enhance the performance in synchronous design. For example, retiming [4] can minimize the worst case clock period by moving in circuit registers. Also Software pipelining was a good approach to generate optimal clocking schemes [5], better that retiming when longest paths between registers cannot be equalized; registers are then activated at non uniform time intervals, like what we propose here. But these methods take in account static worst case delays only, not the delays variations according to dynamic conditions.

Multisynchronous circuits [6] breaks large synchronous chip into multiple submodules that use different phases of the same basic clock. That method resolves, at the system level, only a part of the worst case timing problem by reducing the clock distribution effects.

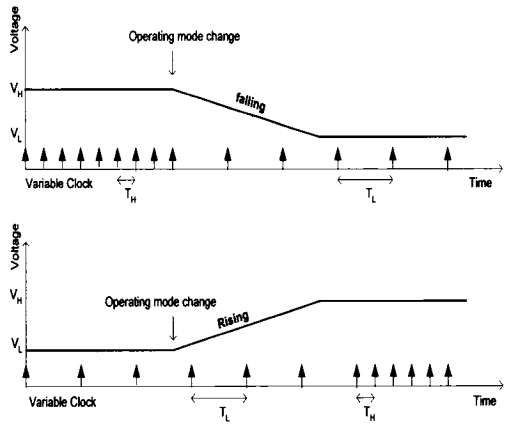

Figure 3. Voltage regulation

We designed a Variable Period Clock Synthesizer (VPCS) that can generate a clock period on a per cycle basis [7], by selecting several equidistant clock phases according to the fraction or multiple of the reference clock period. Here we present an application that shows the ability of a processor to change its clock period using that special clock generator. Our design is also suitable for DVS applications in typical operating conditions; a voltage scheduler, which is an operating system component for use in DVS, can set the voltage and the speed accurately by a defined special instruction.

III. ARCHITECTURE OVERVIEW

The main idea of this work is to set the duration of a particular cycle according to the worst case time the instructions being executed in that cycle can take. Each type of instruction can activate a different critical path, so the cycle duration will change from cycle to cycle. Stating with a pipelined processor, the opcode of each stage is extracted to predict the length of the slowest stage according to the instruction stream. Pipeline control signals are also picked to manage the stalls and set the duration for the right cycle. During design, instruction delays are measured for different voltages and stored in a table which will be used at run-time as the duration for those instructions.

A. Block Diagram

To demonstrate the concept, we based our design on an existing configurable processor with full access to the VHDL code. We used the Altera's Nios [8], running on an FPGA board, configured as a 32 bits data, and 16 bits instructions system, with additional logic added beside the processor core in order to obtain the desired VSP. The signals needed to be extracted from the processor pipeline, and the overview of the added logic to control the clock period is shown in Fig.2. The 'clock' going out of that circuit is used as the clock for the whole system; only the VPCS receives another clock, as a reference clock. As a limitation of the FPGA board used, variable voltage was not possible, so the numbers in the delay memory for low voltage operation are not actual numbers and energy saving could not be measured physically. Also, only 4 phases can be taken from the PLL.

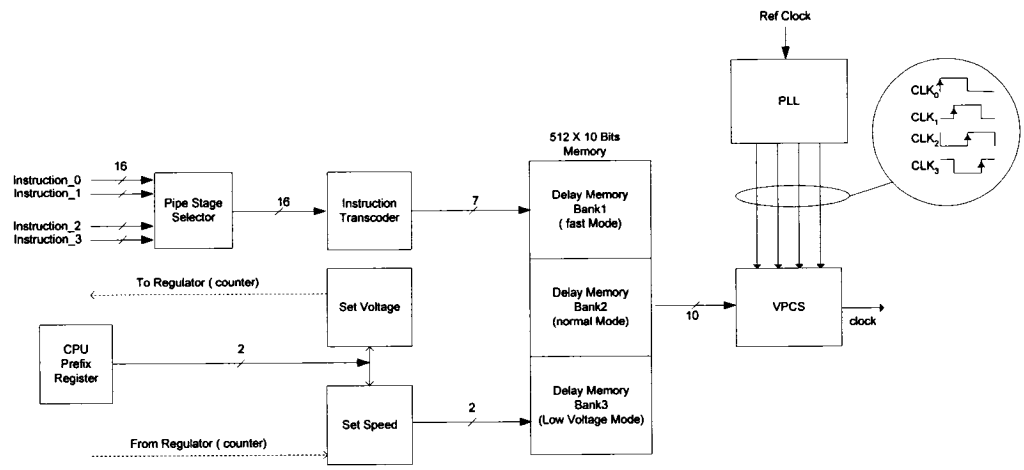

Figure 2. VSP block diagram

B. Control Logic

Four pipeline Stages from the instruction field are available inside the Nios. The different stages instruction opcodes feed the Pipe Stage Selector, which selects the "longest" one to control the period duration.

Since instruction opcodes have different bits width, the *Instruction Transcoder*, transforms the opcodes to a standardized format which will be used, in combination with the current voltage mode from the *Set Speed* module, as address in the *Delay Memory*. This memory contains the delay of each instruction, as a number of base phases, which is sent to the VPCS. For this prototype, the PLL on the FPGA was able to provide only 4 quadratic clock phases to the VPCS; the shortest distance between two consecutive phases had to be above 2ns.

The memory is partitioned in different speed mode banks, each containing the delays information for a specific voltage mode. The *Set Speed* module will select these banks depending on the special instruction *setmode* that is defined as a custom instruction in the configurable CPU. A custom circuitry is added into the CPU ALU to execute that instruction. The special logic we inserted does not change any CPU register value, but picks the 2 LSBs from the prefix register to set the value of the speed mode as well as to control the *setvoltage* module, which output controls the voltage regulator for DVS support.

Voltage regulation can take tens of microseconds to react, so a signal is needed by the *Set Speed* module to know when voltage reaches the specified value. In our prototype, without real DVS, a counter is used as a voltage regulator to show that the functionality is correct. The end of the count then means that voltage regulation is complete.

As shown in Fig.3. to avoid any corruption during the voltage change, the set speed mode logic compares the actual mode with the desired mode and does one of these two actions:

- If the actual speed is slower than the desired one, which means the operating system attempts to raise the voltage and the frequency, the mode will be set after the counter acknowledge the regulation is completed, the target speed is set after the voltage,
- If the actual speed is faster than the desired one, which means the operating system attempts to lower the voltage and the frequency, the speed is set immediately, since the design can run slow even if the voltage has not reached the desired lower value.

C. Pipeline issues

When a stall occurs, signals *commit* and *pipe_run* from the CPU core change to a logical low value. *Commit* signal will change when the instruction being executed requires more than one cycle. Only pipeline stages 1 and 2 can acknowledge this signal. However, *pipe_run* signal can change anytime there is a memory access or a data hazard, creating a pipeline stall. All stages can acknowledge the *pipe_run* signal. We need those signals to change the duration of the right cycle for each operation. For a multi cycle instruction, the *commit* cycle will prolong the selected opcode duration as long as it remains low, as well as the *pipe_run* signal.

Only two types of instructions are confronted for pipeline issues; those that execute into the ALU and the others that access the memory. In this design, the Branch-type instructions are neglected, since the delay consumed for these operations is small compared to the other types. ALU operation logic is used at the fourth stage and memory access is done at the fifth stage. When there is an ALU operation instruction right after a memory access, the 5th pipeline stage of the memory access, and the 4th of the ALU operation happens at the same time. The pipe stage selector will select the longest one to avoid data corruption by checking if the previous or the next instruction to be executed is longer or not. In later section, it is demonstrated that for the Nios the ALU operations have always a longer delay than the memory access.

IV. EXPERIMENTAL RESULTS

The measurements for instruction delays have been made by running timing simulations on a standard Nios processor core obtained after placement and routing. The instruction memory is filled with simple assembly programs. Many simulations with different operating frequencies have been made to verify the patterns between pipeline registers. Since some instruction delays are data dependent, extreme values are used to achieve the longest delays for each instruction. When the timing simulation results do not match the functional simulation results, this is due to a delay constraint violation and we consider smallest period without data corruption.

TABLE I. EXPERIMENTAL MEASUREMENTS
(Same looping program running for 100μs on standard processor and on processor with the added circuit for VSP.)

Metric	Target Processor		
	Standard Nios 10ns cycles	*VSP (1 speed) 10ns cycles*	*VSP (2 speeds) 7.5 & 10ns cycles*
Circuit size (LEs)	2634	2669	2669
Power (W)	1.714	1.777	1.809
Loops/100μS	514	514	602
Loop energy (μJ/Loop)	0.332	0.345	0.299
Speed increase	–	0%	17%
Loop Energy savings	–	-3.7%	9.8%

After testing the different stages with all the instructions, we noted that, for most instructions, the delay of the decode stage is dominating. A floor of 6.5ns has to be used for that reason, and a few instructions are taking 7ns or 7.5ns. These are the *add*, *sub*, *cmp*, *neg*, *swap*, and shifts instructions. This would give a theoretic maximum speedup, associated with our special clocking, of 7.5/6.5 (around 15%), with same energy consumption by the processor core, for this particular processor when no custom instruction is added. This may not seem much, but this is only a proof of concept on a processor which was not originally designed to take advantage of the per cycle clock period adjustment.

Our experimental setup consists of running a custom program that uses both memory access and ALU operations into different processor configurations for the same amount of time. Due to clock tree timing constraints on the FPGA, our VSP is used to produce a clock period of 7.5ns for fast instructions and 10ns for slow instructions. Therefore, the operating frequency on the standard Nios is 100MHz, close to its peak 115MHz frequency. To show the small impact of the added circuitry and the benefit of the cycle by cycle variable clock period, comparative measurements are shown in Table I[G1]. We trigger the Loop energy metric to show the advantageous tradeoff between performance and energy saving. The effective energy per loop in μJ/Loop is calculated as the ratio of the energy consumption to the throughput of the program loop. There is only 3.7% loop energy overhead due to the added circuit for VSP, when used to generate the same clock as on the standard processor. On the other hand, there is a loop energy reduction of 9.8% when using it to generate two different cycle lengths, for short and long instructions.

The speedup of such a design is program dependent. Table I results show a program run time speedup of 17% over the basic Nios, which means that approximately 58% of the cycles were shorter than on the standard processor.

A. What happens with custom instructions?

The Nios is a configurable processor, and a designer can add custom instructions to increase the performance of a desired application. Most of the time, these instructions are complex, to squeeze as much computation as possible from an instruction. Thus the bottleneck in the application becomes the circuit delay and not the instruction set architecture. For this reason, custom instructions may take much more time than a regular instruction, and would thus be better candidates for our special clocking. With our method, the cycle for a custom instruction can be stretched as much as needed, without changing the cycle time of the other instructions. These long cycle times are easily added to the delay memory.

V. CONCLUSION

In this paper, we presented the concept of a VSP that can vary its clock period every cycle according to program instructions, by predicting the worst execution stage in the pipeline. The design is aware of pipeline stalls problems and longer instructions are handled without effort and without slowing down the other instructions. This gives a speedup compared to standard clocked design with little impact on energy consumption, and this speedup can be traded to energy saving by reducing the voltage. This concept can be used in conjunction with dynamic voltage scaling, in order to reduce energy consumption according to the current performance requirement of the application.

A prototype design has been made, based on the Altera Nios configurable processor, running on an FPGA development board. When no custom instructions were used, the speedup was not terrific since the delay was mainly dominated by the decode stage, which is always in use regardless of the instruction. Future research should be made on how a processor could take more advantage of our special clock. As a stating point, critical path

analysis should be viewed differently. In current analysis, the number of critical paths and their length is important. With our clock, the percentage of time a critical path is activated is important, and if the path is not activated then another path must be claimed critical.

Currently, as limited by the development board, the VPCS cannot have a lot of phases, and no dynamic voltage scaling is done. In future research, we should design a very-low power VPCS, so that at low voltage and frequency everything would consume very low energy.

REFERENCES

[1] S.P.Mohant, N. Ranganathan, V. Krishna, "Datapath scheduling using dynamic frequency clocking," in VLSI, 2003 Proceedings of the IEEE Computer Society Annual Symposium, 25-26 April 2002, pp. 58–63.

[2] M. Olivieri, A. Trifiletti, and A. De Gloria, "A Low-Power Microcontroller with On-Chip Self-Tuning Digital Clock Generator for Variable-Load Applications," in Proceedings of the 1999 International Conference on Computer Design: IEEE, 1999.

[3] A. K. Uht, "Uniprocessor Performance Enhancement Through Adaptive Clock Frequency Control," in Proceedings of the SSGRR-2003w International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, e-Medicine, and Mobile Technologies on the Internet. L'Aquila, Italy: Telecom Italia, January 6-12, 2003. Invited paper.

[4] C. Leiserson and J. Saxe, "Retiming Synchronous Circuitry," Algorithmica, vol. 1, 1991, pp. 3–35.

[5] F. R. Boyer, E. M. Aboulhamid, Y. Savaria, and M. Boyer, "Optimal Design of Synchronous Circuits Using Software Pipelining Techniques," ACM Transactions on Design Automation of Electronic Systems, vol.6, no.4, 2001, pp. 516–532.

[6] R. Ginosar and R. Kol, "Adaptive Synchronization," in Proceedings of the 1998 International Conference on Computer Design, 1998.

[7] F.R. Boyer, H.G. Epassa, B. Pontikakis, Y. Savaria, "A Variable Period Clock Synthesis (VPCS) Architecture for Next-Generation Power-Aware SoC Applications", in Proceedings of the 2nd International IEEE Northeast Workshop on Circuits and systems, 2004, pp

[8] Altera Corp. www.altera.com